mi computer

CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

TOMO 6

Editorial Delta, S.A.


mi COMPUTER

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, S. Tarditti, F. Martín
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

ISBN: 84-7598-111-9 (tomo 6) 84-85822-82-X (obra completa)
Depósito legal: B. 52/1984
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona)
Impreso en España - Printed in Spain - 1985

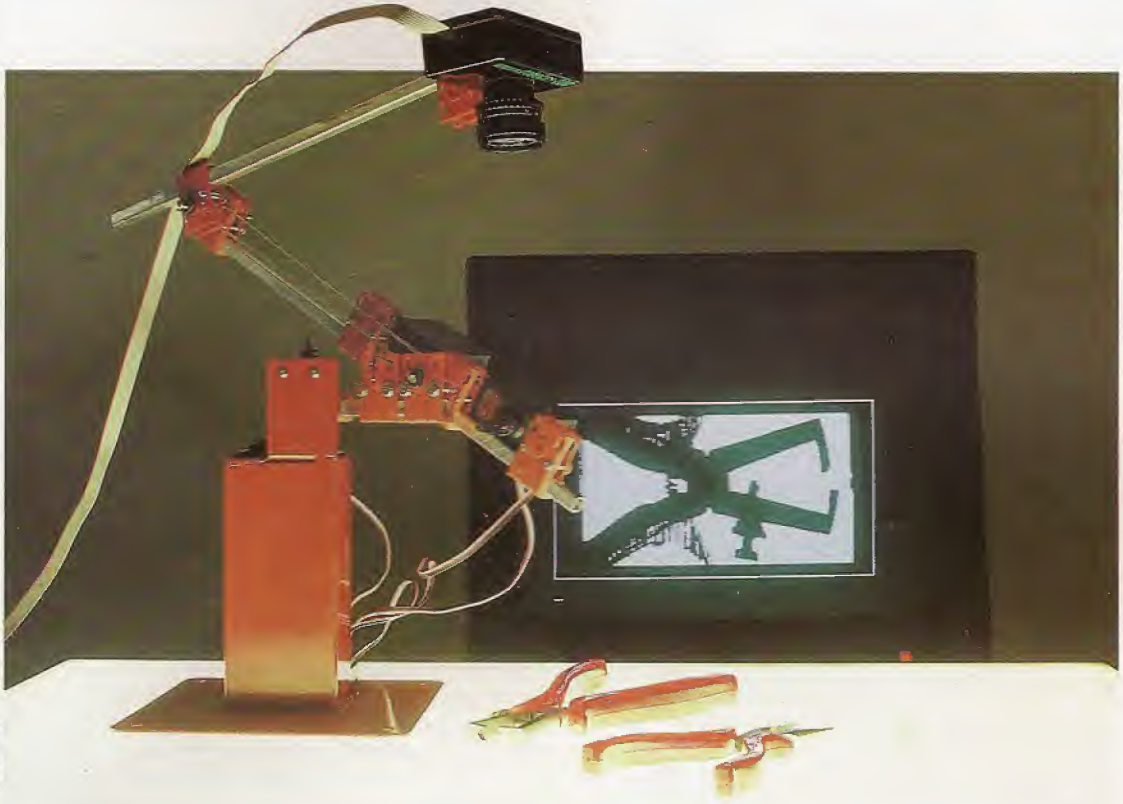
mi COMPUTER

VOLUMEN 6

Editorial Delta, S.A.

Superar obstáculos

Ha llegado el momento de que estudiemos cómo conseguir que un robot se mueva de una manera auténticamente "inteligente"



Ian McKinnell

Construyendo una imagen
Si sus procesadores son suficientemente sofisticados, los robots móviles pueden aprender un catálogo de objetos arquetípicos para utilizar con algoritmos para reconocimiento de formas y comparación de patrones. El brazo-robot Beasty, que vemos en la fotografía, está equipado con una cámara Snap, que produce una imagen digital, y el software Snap, que incluye un módulo para reconocimiento de objetos. Una vez que el objeto ha sido "visto" desde distintos ángulos, el brazo tiene una razonable posibilidad de reconocerlo en cualquier posición

En primer lugar, no queremos controlar al robot mediante un operador humano. Si el operador debe observarlo y controlar hasta el más mínimo de sus movimientos, entonces en muchas aplicaciones la utilización de un robot no tendría el menor sentido: la persona bien podría llevar a cabo las tareas que realiza éste. Esto, por supuesto, no es así para todas las situaciones. Los robots que se utilizan para desactivar bombas son controlados por el hombre, porque para guiarlos correctamente aún sigue siendo necesaria la experiencia humana.

Asimismo, tampoco tiene mucho sentido controlar un robot a través de una secuencia fija de instrucciones almacenadas en un ordenador. Esto resultaría ser poco más que un autómata, un dispositivo que sigue al pie de la letra la secuencia incorporada, independientemente de las circunstancias. Nuevamente hay ocasiones en la que tales dispositivos son útiles: con frecuencia los brazos-robot se consideran "inteligentes" aun cuando lleven a cabo un conjunto preprogramado de acciones.

Sin embargo, hemos definido al robot "inteligente" como aquel que pueda servir una taza de café por la mañana. Este robot no podría ser controlado por un ser humano, dado que su función consiste en llevar a cabo su tarea cuando aún no hay ningún ser humano despierto. Si este dispositivo servidor de café se programara con una secuencia fija de instrucciones, surgirían problemas si se cambiara de lugar la cama o se dejara ropa en el suelo.

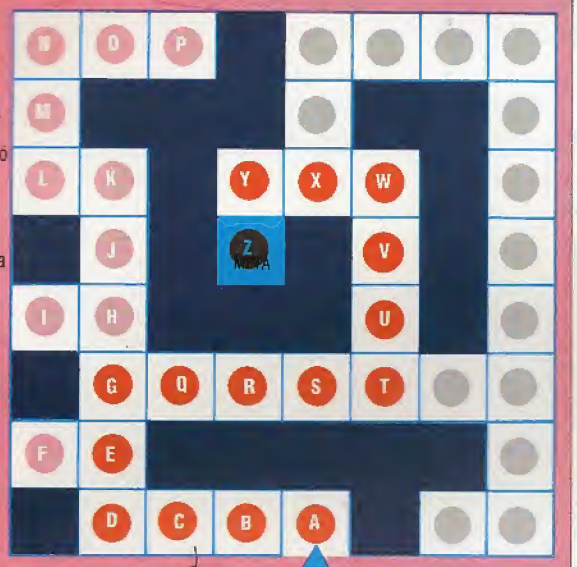
De modo que nuestra definición de movimiento inteligente es la capacidad de un robot de desplazarse por su entorno sin que lo controle ningún ser humano y sin seguir ciegamente ninguna secuencia fija de instrucciones. Debería ser capaz de moverse de un punto a otro, evitando cualquier obstáculo.

En el campo de la inteligencia artificial existe la tradición de utilizar juegos de una u otra clase para examinar problemas complejos de esta clase. Así como los programas para jugar al ajedrez han permitido profundizar considerablemente en otras ramas de la inteligencia artificial, los robots que corren a través de laberintos pueden ayudarnos en nuestra definición de movimiento auténticamente inteligente. A fines de los años setenta comenzaron en Estados Unidos las competiciones de "microrratones", y en 1980 se celebró en Gran Bretaña la primera competición de esta naturaleza. La idea era muy sencilla: se construyó un gran laberinto de alrededor de tres metros cuadrados y los participantes hubieron de diseñar "ratones-robot" que pudieran descubrir por sí mismos el camino hasta el centro sin ninguna clase de ayuda. El laberinto se componía de pequeños cuadrados de igual tamaño cuyos lados en ocasiones estaban abiertos para mostrar una posible ruta y otras veces cerrados

Kevin Jones

Tan sencillo como el ABC

En la resolución de laberintos funcionan algoritmos simples. Este robot avanza por espacios vacíos hasta que llega a un callejón sin salida, los cuadrados F, I y P, por ejemplo. Entonces retrocede hasta la última intersección que encontró en este caso, el cuadrado G) marcando todos los cuadrados recorridos como inútiles en su mapa mental. Si desde una intersección no hay ninguna ruta todavía no probada, el robot retrocede hasta la intersección anterior a ésta y así sucesivamente, todo el camino de regreso hasta la entrada, si fuera necesario, en cuyo caso el laberinto es "ciego" o insoluble



Solución al laberinto

```
 49 REM·····CBM 64··········
 50 REM·SOLUCION AL LABERINTO·
 51 REM·····CBM 64··········
100 GOSUB 2000              :REM INICIO
150 GOSUB 9000              :REM IMPR.LABERINTO
200 FOR L=1 TO 1
250 GOSUB 7000              :REM MIRAR ALREDEDOR
300 GOSUB 8000              :REM MOVERSE
400 NEXT L
450 FI=16:CO=12:GOSUB 9600
500 IF LAB < > HM THEN PRINT"CIEGO"
550 IF LAB=HM THEN PRINT"CASA"
900 PRINT:PRINT:INPUT A$:RUN
1999 REM ·····················
2000 REM ·          INICIO          ·
2001 REM ·····················
2100 TM=10:S2=TM·TM:GX=TM/2:GY=TM+2
2120 DIM LAB(TM,TM),R$(4),LX(4),LY(4)
2140 X=RND(-TI)
2150 DEFFNR(N)=INT(RND(1)·N+1)
2160 HM=-1:G0=-2:WL=42:WS=CHR$(WL)
2180 CL$=CHR$(147):H$=CHR$(19)
2200 X=GX-1:Y=GY-2:DR=3
2220 D$=CHR$(17):P$=D$
2240 FOR K=1 TO 5:P$=P$+P$:NEXT K:P$=H$+P$
3400 DATA 0,1,"0",-1,0,".",0,-1," ",1,0," "
3420 FOR K=1 TO 4:READ LX(K),LY(K),R$(K)
3480 NEXT K:RETURN
5999 REM ·····················
7000 REM ·    MIRAR ALREDEDOR    ·
7001 REM ·····················
7110 FI=1:CO=1:GOSUB9500
7120 L=0:ND=0:FOR S=1 TO 4
7140 NX=X-LX(S):NY=Y+LY(S)
7200 IF NX < 1 OR NX > TM THEN NEXT S:RETURN
7220 IF NY < 1 OR NY > TM THEN NEXT S:RETURN
7230 LAB=LAB(NX,NY):IF LAB=0 THEN ND=S
7250 IF LAB=HM THEN ND=S:S=4:L=1
7280 NEXT S:RETURN
7999 REM·····················
8000 REM· MOVERSE           ·
8001 REM·····················
8100 LAB(X,Y)=DR:FL=1
8110 FI=FI+1:CO=X+1:GOSUB 9500
8140 IF ND=0 THEN ND=DR-2-4·(DR<3):FL=2
8150 PRINT R$(FL);:X=X+LX(ND):Y=Y+LY(ND)
8160 DR=ND: IF Y>TM THEN L=1:RETURN
8200 IF FL=2 THEN DR=LAB (X,Y)
8400 RETURN
8998 REM ·····················
8999 REM ·   IMPRIMIR LABERINTO   ·
9001 REM ·····················
9040 PRINT CLS:FI=1:CO=1
9050 WL=42:WS=CHR$(WL)
9060 S$=
9070 FOR K=1 TO TM-2:T$=T$+W$:NEXT K
9080 B$=W$+LEFT$(S$,TM)+W$
9100 PRINT B$:FOR J=2 TO TM+1
9120 PRINT B$:NEXT J:PRINT T$
9140 FOR K=1 TO SQ:2
9150 NR=FNR(TM):NY=FNR(TM)
9160 LAB(NX,NY)=WL:FI=WY+1:CO=WX+1
9200 GOSUB 9500:PRINT W$:NEXT K
9250 CO=1+FNR(TM):FI=1+FNR(TM):GOSUB 9500
9300 PRINT"H";LAB(CO-1,FI-1)=HM
9350 FI=FI+2:CO=CO:GOSUB9500:PRINT"·"
9400 RETURN
9498 REM·····················
9500 REM·  POSICIONAR CURSOR  ·
9501 REM·····················
9502 PRINT LEFT$(P$,FI);TAB(CO-1);:RETURN
```

Complem. al BASIC

Introduzca las siguientes modificaciones en este programa:

BBC Micro

```
 49 REM ··············· BBC ···············
 50 REM ·       SOLUCION AL LABERINTO      ·
 51 REM ··············· BBC ···············
9040 CLS:FI=1:CO=1
9600 PRINT TAB(CO-1,FI-1);:RETURN
```

Spectrum

```
 49 REM ·············· SPECTRUM ··············
 50 REM ·       SOLUCION AL LABERINTO      ·
 51 REM ·············· SPECTRUM ··············
2120 DIM LAB(TM,TM):DIM R$(4)
2130 DIM LX(4):DIM LY(4)
2140 RANDOMIZE
2150 DEFFNR(N)=INT(RND·N+1)
9040 CLS:FI=1:CO=1
9600 PRINT AT (FI-1,CO-1);:RETURN
```

Brian Morris

para denotar una pared. El ratón que llegó al centro en el menor tiempo ganó la competición.

En el primer certamen de microrratones celebrado en Gran Bretaña sólo hubo cinco participantes. Algunos se comportaban de una forma sumamente errática; uno ni siquiera podía avanzar en línea recta y hasta el mejor de los ratones quedaba bastante confundido después de dar la vuelta a un par de esquinas. En el mismo año se celebraron las finales europeas de esta competición y participaron ratones procedentes de Finlandia, Suiza y Alemania. Finalmente hubo uno que consiguió recorrer el laberinto correctamente; se trataba del *Stirling mouse* de Nick Smith, que estaba equipado con sencillos sensores mecánicos que corrían a lo largo de la parte superior de las paredes del laberinto y funcionaba con un sencillo motor paso a paso. Desde entonces, el interés en esta clase de competiciones ha aumentado, y en la Euromouse Contest que se celebró en Madrid en 1984 el mejor tiempo conseguido para alcanzar el centro del laberinto fue de 31,4 segundos. Algunos participantes tampoco entonces pudieron llegar a la meta, pero la mayoría sí lo logró.

El mapa del laberinto

¿Y cómo recorre un ratón-robot un laberinto? En general, el robot debe usar un método preciso para su desplazamiento, de modo que sepa exactamente cuál es su posición en cualquier momento dado; esto se puede lograr montando al robot sobre ruedas y activándolo con motores paso a paso, a menudo utilizando alguna forma de realimentación interna, como codificadores de eje. El robot requiere, asimismo, un conjunto de sensores para detectar la presencia o la ausencia de paredes, lo cual le permite construir un "mapa" del laberinto. En las competiciones de microrratones a los robots se les concede un par de vueltas de entrenamiento, que utilizan para elaborar un plan del recorrido. Luego efectúan la vuelta de competición, durante la cual se cronometran sus intentos por alcanzar el centro.

Si bien los métodos exactos varían de un robot a otro, una respuesta es equipar por delante al robot con un sencillo detector táctil. Colocado en el centro de cada cuadrado del laberinto, puede verificar si tiene alguna pared exactamente delante suyo. Luego gira 90° en el sentido de las agujas del reloj, vuelve a verificar y repite la secuencia. Finalmente "sabrá" dónde están todas las paredes de cada cuadrado del laberinto. Esta información se puede almacenar como un único número binario de cuatro bits; de modo que 1111 en binario representaría un cuadrado con paredes en sus cuatro lados (imposible en la práctica, porque en ese cuadrado el robot ni siquiera podría entrar), y 0000 correspondería a un cuadrado sin ninguna pared. 0111 representaría entonces un cuadrado con tres paredes y una abertura: un callejón sin salida.

Esta información se podría retener en una matriz bidimensional; en BASIC se podría utilizar DIM A (16,16) para representar un laberinto con 16 "celdas" en cada dirección. El robot entonces ha de calcular una ruta que lo conduzca hasta A(8,8), si se considera que ése es el centro del laberinto. A menudo el robot posee un programa de ordenador incorporado que calcula una estructura de árbol para cada ruta a través del laberinto. Muchas de las ramas del árbol conducirán a callejones sin salida o

llevarán al robot de regreso hasta un punto que ya ha visitado; en estos casos las bifurcaciones se "podan" y no se tienen en cuenta. El programa busca entonces en las bifurcaciones restantes para hallar la ruta con menos cuadrados. Entonces adopta ese camino como su ruta hacia el centro.

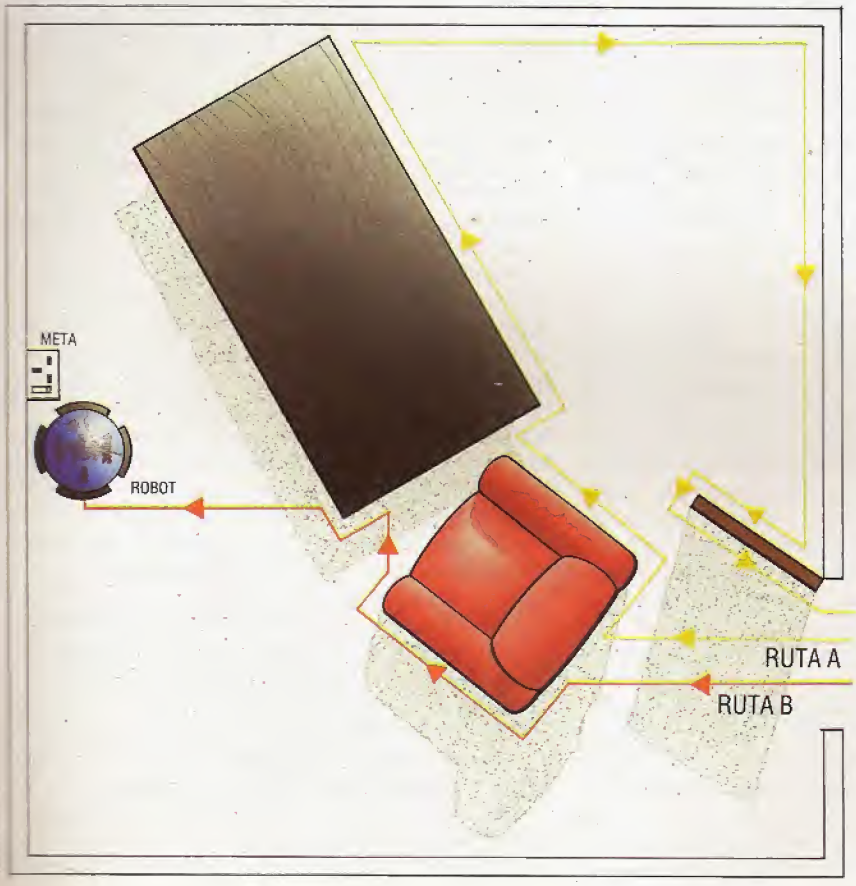
Se puede adaptar este método para que proporcione una estrategia más eficaz. Los sensores del robot son esenciales para su éxito. Por ejemplo, sensores táctiles mecánicos sencillos exigen que el robot choque realmente contra cada una de las paredes para trazar su mapa; los sensores de proximidades pueden detectar una pared sin tocarla y un sensor de distancia puede detectar la posición de una pared al final de un camino largo y despejado a través del laberinto. Obviamente, equipar al robot con cuatro sensores en vez de uno le permitiría "mirar" en las cuatro direcciones a la vez y eliminaría la necesidad de hacerlo girar en cada cuadrado.

Por la casa

De modo que vemos que un robot puede actuar "inteligentemente" cuando recorre un laberinto. En muchos sentidos, el problema de construir un robot que pueda hallar su camino a través de su casa es muy similar. El robot debe utilizar sensores para calcular las posiciones de todos los objetos de una habitación, y debe luego planificar una ruta que lo lleve, sorteando todos los obstáculos, hasta su punto de destino. Los problemas adicionales que implica este tipo de movimiento inteligente provienen del hecho de que el diseño de una habitación es muchísimo más complejo que el diseño de un laberinto. Una habitación corriente no está dividida simplemente en cuadrados, ni todos sus componentes permanecen fijos en el mismo lugar. El robot servidor de café podría aprender la posición de varios objetos; pero si se moviera una silla o si el gato se sentara en el suelo, el robot debería entonces modificar el camino elegido.

Este problema sólo se puede resolver haciendo que el robot haga un uso continuo de sus sensores para actualizar su mapa interno. El problema del gato exige mayor atención porque, dado que los robots no saben nada de gatos (como tampoco, si vamos al caso, de personas), al robot le resulta difícil calcular lo que debe hacer en su primer encuentro con un felino. (Qué duda cabe de que el gato tendrá el mismo problema cuando se encuentre por primera vez con un robot.) La mejor solución es equipar al robot con un sensor de movimiento, que es un sensor de distancia que responde a la medición de distancias variables y, por consiguiente, puede hacer frente a objetos en movimiento. Una vez detectado algo que se está desplazando, lo mejor que puede hacer el robot es permanecer inmóvil hasta que el objeto deje de moverse o se aleje. Puede ser que esto no parezca muy inteligente, y por cierto es menos cordial que acercarse al gato y acariciarlo, pero una acción de este tipo es muy similar a la reacción de muchos animales, que se quedan estáticos cuando detectan objetos en movimiento.

El tema del movimiento inteligente va, por consiguiente, ligado al empleo de sensores conjuntamente con un programa de ordenador. Un robot sin sensores no será capaz de moverse inteligentemente y, mientras más sensores tenga instalados, mejor será su conocimiento del mundo. Es este conocimiento lo que le permite al robot mostrar signos de inteligencia.



Por dos caminos
Abrirse paso a través de objetos desconocidos nunca es fácil. La ruta A muestra la huella de un sencillo robot doméstico tratando de hallar el enchufe de la corriente. Su único algoritmo para evitar objetos (conocido como *seguimiento de pared*) es seguir los bordes de las cosas mientras sus sensores de corto alcance buscan el objeto. Este método primitivo puede ser muy eficiente en entornos sencillos, pero trampas y escollos como los que vemos en la ilustración pueden hacerlo fracasar.
La ruta B muestra la huella de un robot similar con un algoritmo ligeramente mejorado: cuando ha de girar un cierto ángulo al hallar un objeto, prefiere hacerlo describiendo el menor ángulo posible, dado que ello reducirá la cantidad de "rastreo hacia atrás" que realice. Esta sencilla modificación reduce en gran medida su vulnerabilidad a las trampas y permite que sus sensores de exploración controlen su comportamiento con mayor eficacia

Programa modelo

Esta vez analizaremos el "Abacus", el paquete de hoja electrónica que se suministra con el Sinclair QL

MENÚ DE MENSAJES

MENÚ DE INSTRUCCIONES

Gran parte del interés que generó el Sinclair QL se ha centrado en los cuatro paquetes de software que se suministran junto con la máquina: *Quill* (tratamiento de textos), *Archive* (base de datos), *Easel* (programa para gráficos) y *Abacus* (hoja electrónica). Estos paquetes contienen algunos elementos de diseño integrado (véanse pp. 982-983). Se puede transferir datos entre ellos; los modelos de hoja electrónica, por ejemplo, se pueden visualizar como gráficos o incorporar en un documento preparado mediante *Quill*. Las pantallas de visualización son similares y algunas instrucciones son comunes a todos los paquetes: por ejemplo, tres de las cinco teclas de función del QL dan resultados idénticos en las cuatro aplicaciones (F1 es la tecla Help, F2 controla la zona de "mensajes" de la parte superior de la pantalla y F3 llama a las instrucciones). No obstante, los programas se deben cargar y ejecutar por separado.

Hay dos formas diferentes de cargar el *Abacus* (de hecho, cualquiera de los paquetes del QL). La primera supone colocar el cartucho en el microdrive 1 y después pulsar F1 para seleccionar la opción pantalla o F2 si se utiliza un televisor para la visualización. Los paquetes del QL incluyen rutinas para la carga automática de los programas *(boot)*. Por el contrario, si ya se ha seleccionado la pantalla, entre lrun mdv1_boot (suponiendo que *Abacus* esté en la unidad 1) y aparecerá la pantalla inicial.

La pantalla muestra la porción superior izquierda de la matriz de la hoja electrónica; en su documentación, Psion, la firma fabricante, alude a la misma como una "cuadrícula". Inicialmente, se visualizan las columnas de la A a la F y las filas del 1 al 15, si bien las dimensiones máximas de la cuadrícula del *Abacus* son de 64 columnas y 255 filas. (Compare estas dimensiones con las dimensiones máximas de la cuadrícula del *Vu-Calc*, de 28 columnas y 55 filas.) En la parte superior de la visualización de la cuadrícula hay un conjunto de mensajes, y debajo de ella hay una línea para entrada de datos, junto con información que refleja el estado del modelo en curso. Los mensajes se pueden eliminar (pulsando F2), pero son muy útiles para los principiantes, dado que explican en cada momento las opciones disponibles. Ésta es, en realidad, una zona de "menú" que indica qué teclas de función se utilizan para controlar diversas operaciones y que muestra cómo mover el cursor o ir directamente hasta una celda determinada, cómo entrar datos o texto y cómo llamar a las instrucciones. Sin embargo, no es un auténtico menú: las opciones no se pueden seleccionar posicionando el cursor sobre la opción correspondiente, sino que el usuario las debe digitar.

La utilización del *Abacus* para tareas básicas es muy simple, si bien para un tipo de tareas más avanzado llevará un tiempo acostumbrarse a algunas instrucciones y expresiones. El siguiente ejemplo, basado nuevamente en un presupuesto doméstico, ilustrará la forma de trabajar del *Abacus*.

En primer lugar, necesitamos un encabezamiento general. Al igual que con el *Vu-Calc*, las entradas de textos deben ir precedidas por comillas. A nuestro modelo lo llamaremos PREVISIÓN DE EFECTIVO, y pulsando la tecla de cursor apropiada, pasamos a la celda D1 y simplemente entramos unas comillas seguidas del texto. El *Abacus*, al igual que la mayoría de las hojas electrónicas, permite que el texto exceda de una celda si la adyacente está vacía, de modo que es fácil entrar hasta los títulos más largos.

También podemos subrayar el encabezamiento, mejorando de ese modo el aspecto de nuestro modelo. Para hacerlo debemos desplazar el cursor hasta la celda de debajo de nuestro título (D2) y entrar rept ("=",len(d1)). Aquí rept es el equivalente de la instrucción REPLICATE del *Vu-Calc*, y = le dice al programa qué símbolo utilizar (estamos utilizando el signo "igual" como un subrayado doble). El resto de la instrucción (len(d1)) es la forma de decirle a *Abacus* que repita el símbolo para toda la longitud del texto de la celda D1.

A diferencia del *Vu-Calc*, que posee una anchura de columna fija de nueve caracteres, *Abacus* nos permite seleccionar anchuras distintas para diferentes aplicaciones. En nuestro caso necesitamos que la columna A sea más ancha, para que haya espacio suficiente para entrar texto de longitud variable, y además precisamos que las otras columnas sean más estrechas, de modo que se puedan visualizar los datos para seis meses. Para hacer esto, utilizamos la tecla de función F3, seguida de G (para seleccionar la instrucción GRID, cuadrícula) y W (para la instrucción WIDTH, anchura). La línea de entrada indicará que la anchura actual es 10. Nosotros deseamos cambiar la anchura de la columna A a 15, de modo que tecleamos 15 en respuesta al mensaje. El programa ahora nos solicitará una serie de celdas a las cuales se aplicará la nueva anchura. Entrando A y A como los dos parámetros indicamos que sólo la columna A debe poseer esta anchura. Luego debemos volver a pasar por el mismo procedimiento, esta vez seleccionando una anchura de 6 y una serie de B a G. Con esto obtenemos lugar suficiente para visualizar las cifras de seis meses.

Ahora hemos de entrar etiquetas para las columnas de "meses". Ello se puede hacer simplemente digitando el texto correspondiente en cada celda, pero *Abacus* posee una facilidad especial para los nombres de los meses. Desplace el cursor hasta A3 y digite row=month(col()−1) (fila=mes...). La línea de entrada solicitará ahora una serie entre B y G en respuesta a los mensajes, y las columnas se etiquetarán automáticamente. Hay nombres de meses demasiado largos como para caber en nuestras columnas ajustadas, por lo cual hay que abreviarlos. Esto

se realiza volviendo a digitarlos, recordando colocar las comillas para indicar texto.

El siguiente paso consiste en entrar encabezamientos para las diversas filas, desplazando el cursor hacia abajo una línea después de cada entrada (lamentablemente, *Abacus* no lo proporciona como una facilidad automática). Nuestro modelo da por supuesto que el cabeza de familia es un vendedor cuyos ingresos se componen tanto de un salario básico como de comisiones. El modelo permite calcular los ingresos en base a comisiones sobre las ventas previstas más el salario básico. Las ventas reales conseguidas se pueden entrar a medida que vayan transcurriendo los meses, y se pueden entonces realizar previsiones revisadas para los meses futuros. Los valores negativos se visualizan entre paréntesis en la cuadrícula terminada. Los encabezamientos a entrar son éstos: Ventas: previstas y reales; Comisión; Salario básico; Ingresos totales; Gastos: hipoteca, impuestos, agua, electricidad, gas, teléfono; Gastos totales; Líquido positivo (o negativo); Balance bancario de apertura y cierre.

Después de haber entrado los encabezamientos de columnas y filas, podemos empezar a rellenar con cifras nuestra hoja electrónica. Primero entraremos las previsiones de ventas para los seis meses visualizados en la pantalla. Éstas pueden ser, por ejemplo, 200 000, 240 000, 260 000, 220 000, 240 000 y 300 000 ptas, y se deben entrar sin el punto. Las ventas reales no se pueden entrar todavía, de modo que procederemos calculando la comisión sobre las ventas previstas. La misma se calcula como el 20 % de las ventas que superen las 200 000 ptas, de modo que la fórmula a entrar en la celda B10 es row=(ventas−200000)*.2. Nuevamente la serie requerida es de B a G; entre usted estas letras en respuesta a los mensajes y la comisión se calculará y se visualizará al instante.

Si el salario básico es de 34 000 ptas mensuales, este se puede entrar en la celda B11 como row=34000, otra vez para la serie de B a G. Ahora se puede digitar la fórmula para ingresos totales en B13. La misma es row=sum(col) con una serie de filas de 0 a 11 y una serie de columnas de B a G. Esto completa el cálculo de los ingresos, pero se puede mejorar la presentación del modelo agregando rayas arriba y abajo de la fila de "ingresos totales". Para hacer esto, desplace el cursor hasta B12 y entre row=rept("−",width()−2). *Abacus* responderá produciendo cuatro guiones debajo de cada cifra de "salario básico". Para llevar a cabo la misma operación en la fila 14, produciendo por consiguiente guiones arriba y abajo de la fila de "ingresos totales", se utiliza la instrucción ECHO(eco). Con el cursor aún en B9, el sistema pedirá una serie sobre la cual reproducir el subrayado: responda con B14:G14. (Si se olvidó de dejar filas libres cuando entró los encabezamientos, éstas se pueden insertar empleando la instrucción GRID.) Para completar el formateado, se debe "justificar por la derecha" todo el subrayado utilizando la instrucción J para la serie de B2 a G14. Ello asegurará que todos los guiones queden alineados con las cifras.

Ahora ya se pueden entrar los gastos, utilizando la fórmula de fila (*row*) para cifras mensuales estándares tales como la hipoteca, o simplemente entrando cada cifra en las celdas correspondientes para pagos irregulares, como pueden ser la electricidad y el gas. Los gastos totales se pueden, entonces, definir como la suma de todas estas cifras, de la misma forma que se calcularon los ingresos totales.

Del mismo modo, la cantidad líquida positiva o negativa se puede calcular como los ingresos totales menos los gastos totales. Lamentablemente, *Abacus* parece ignorar cualquier parte de una instrucción después de un espacio, de modo que no podemos entrar simplemente row=ingresos totales−gastos totales. Debemos, por consiguiente, volver a designar la línea de ingresos totales como "ingresos" y entrar row=ingresos−total, nuevamente para la serie de B a G. Ahora aparecerán en la pantalla las entradas y las salidas líquidas para cada mes.

El paso final consiste en calcular los balances bancarios. Entre primero el balance inicial: un descubierto de 50 000 ptas, por ejemplo. Entre esta cantidad como −50000. Ahora calcule el balance de cierre, utilizando la fórmula row=neto+apertura con el cursor en B28, para la ya familiar serie de B a G. Ello producirá el balance de cierre de enero. Los balances de apertura para los otros meses se calculan entrando en C27 la fórmula row=B28. Para hacer este trabajo para todas las columnas, primero es necesario cambiar el orden de cálculo de fila a columna utilizando la instrucción DESIGN. Entonces se calcularán todos los balances de apertura y cierre y se volverán a calcular de inmediato si se modificara cualquier cifra. Tal como está, nuestro modelo refleja los balances negativos como cifras precedidas por un signo menos. Para cambiar este formato de modo que una cifra negativa se indique mediante paréntesis, utilizamos la instrucción UNITS y respondemos al mensaje con B.

Ahora nuestro modelo ya está completo. Se puede salvar seleccionando la instrucción SAVE con un nombre de archivo adecuado. Después se puede imprimir o se pueden entrar cifras distintas.

Ian McKinnell

Difundiendo la palabra
Un atractivo diseño de pantalla, buenas características y una amplia facilidad de mensajes/ayudas hacen que el *Abacus* sea una hoja electrónica agradable y fácil de utilizar. Aquí mostramos la instrucción ECHO en acción copiando una fila a otra, las opciones de la instrucción DESIGN para formatear la hoja electrónica y la facilidad *month* (meses), que genera los nombres de los meses con la sola pulsación de una tecla. También se puede apreciar la pantalla por defecto con el menú de mensajes (*prompts*) arriba, y la hoja en blanco mostrando el menú de instrucciones (*commands*)

En busca de nombres

Ya estamos en condiciones de ampliar la utilidad de búsqueda de variables para que incluya la facilidad de cambio de nombres

Nuestro programa para reemplazar variables es una utilidad más compleja que la simple búsqueda de nombres de variables que desarrollamos en las páginas 1144 y 1180. Por este motivo necesitamos añadir un programa en código máquina. La CPU 6502 del BBC Micro y la CPU 6510 del Commodore 64 poseen el mismo lenguaje assembly, de modo que es una buena idea analizarlas juntas.

Nuestra primera tarea consiste en encontrar un método de retener dos programas separados en la memoria del ordenador al mismo tiempo. Como ya hemos explicado, en el BBC Micro podemos hacerlo alterando las variables incorporadas en BASIC PAGE e HIMEM. En el Commodore 64 necesitamos un programa en código máquina para alterar los diversos apuntadores de la memoria de página cero. La primera parte del listado en lenguaje assembly, que comienza en la etiqueta SWITCH, hará esto por nosotros.

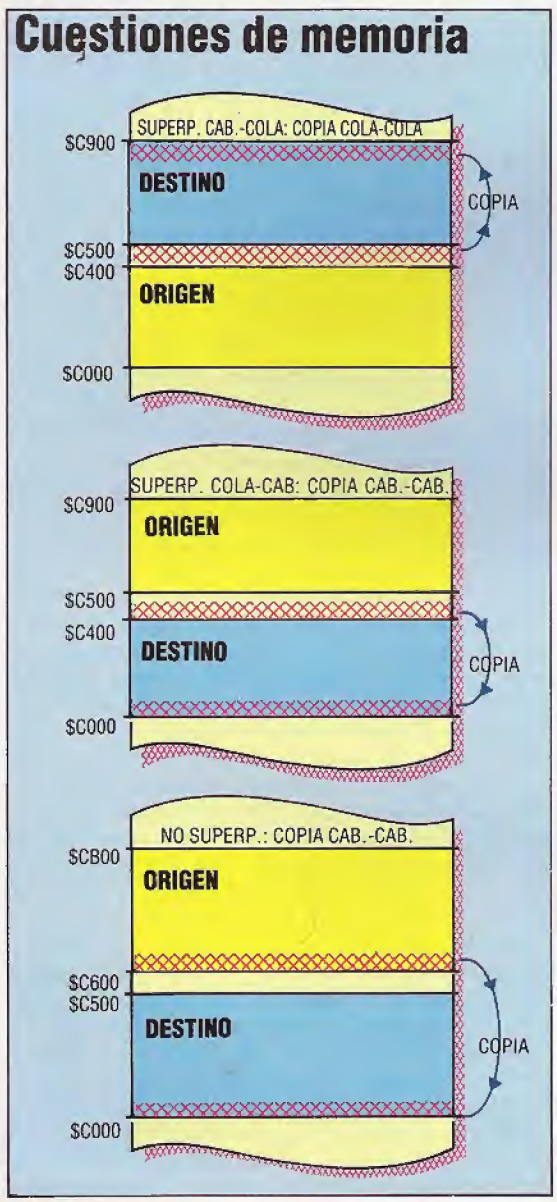
La rutina SWITCH nos permitirá acomodar los programas en BASIC: uno empieza en la dirección 800 hexa (el sitio habitual para un programa en BASIC) y el otro comienza en la dirección 9000 hexa. SWITCH comienza por mirar el apuntador TXTTAB para ver cuál de las zonas de programa es la actual, y cambia entonces los valores del apuntador para convertir en actual la otra zona de programa.

TXTTAB se cambia para que apunte al principio de la nueva zona de programa, luego FRETOP y MEMSIZ deben apuntar al byte que sigue al último byte de la nueva zona de programa, mientras FRESPC apunta al final de la nueva zona de programa. El programa busca entonces en la cadena de apuntadores de direcciones de enlace (véase p. 1184) para hallar el final del programa en BASIC, utilizando VARTAB como apuntador temporal. Cuando encuentra los dos bytes de dirección de enlace a cero que indican el final del programa, incrementa dos veces el apuntador anterior y copia el resultado en ARYTAB y STREND. De este modo, tanto VARTAB como ARYTAB y STREND apuntan al byte que sigue inmediatamente después al programa en BASIC.

Los principales cambios que hemos de introducir en el programa en BASIC son las subrutinas extras de las líneas 30500 y 30600. La primera de éstas encuentra el final del programa en BASIC, utilizando los bytes de longitud de línea en la versión para el BBC y los apuntadores de la siguiente línea en la versión para el Commodore 64.

El cambio en los nombres de variables lo efectúa realmente la subrutina de la línea 30600. Cuando el nombre antiguo de la variable y el nombre nuevo tienen la misma longitud, el nuevo nombre simplemente se puede escribir encima del antiguo. Cuando la longitud del nombre antiguo y la del nombre nuevo son distintas, el procedimiento es algo más complicado. En este caso, el programa debe o bien hacer espacio extra, o reducir todo el espacio innecesario en el programa que esté modificando, y efectuar las modificaciones correspondientes en las variables que utilice para llevar el registro de su posición en el programa que está alterando. Asimismo, debe cambiar el byte de longitud de línea en la versión para el BBC y los apuntadores a la línea siguiente en la versión para el Commodore 64.

El programa también emplea subrutinas en código máquina para crear o suprimir espacio. Aunque esto se podría hacer en BASIC, sería inaceptablemente lento incluso en un programa de tamaño



Liz Dixon

Copia inteligente
La rutina de sustitución ha de desplazar grandes secciones del programa en BASIC hacia arriba y hacia abajo de la memoria cuando inserta nombres de variables de diferente longitud. Al hacerlo satisface las cuatro posibles condiciones de las direcciones de origen y de destino. Si siempre copiara desde el principio del bloque origen, entonces, cuando la cabeza del bloque de destino se superpusiera con la cola del bloque origen, lo que copiara se escribiría encima de algunos de los datos del origen. Una rutina de copia "inteligente" detectará este caso y evitará la corrupción copiando este bloque origen a partir de la cola. Una rutina de copia "tonta" siempre copia a partir de la cabeza del bloque origen

medio, con millares de bytes a desplazar cada vez.

Hay dos subrutinas en lenguaje máquina: UP para hacer espacio extra y DOWN para suprimir el espacio innecesario. Los detalles del bloque de programa a desplazar se pasan al código máquina a través de las posiciones de memoria CURR, LAST y DIFF.

Para la subrutina UP es necesario inicializar las siguientes posiciones de memoria:

CURR: dirección de la parte inferior del bloque a desplazar
LAST: uno menos que la dirección de la parte superior del bloque a desplazar
DIFF: cantidad de bytes a liberar

y para DOWN es necesario inicializarlas así:

CURR: dirección de la parte superior del bloque a desplazar
LAST: uno menos que la dirección de la parte inferior del bloque a desplazar
DIFF: cantidad de bytes a obtener

Observe que las dos subrutinas empiezan en extremos opuestos del bloque a desplazar y realizan sus desplazamientos en sentidos contrarios. Esto es para evitar que los datos se sobreescriban antes de haberlos desplazado.

Para utilizar la versión BBC del programa de sustitución de variables, primero debe entrarse (o cargarse, LOAD) y luego ejecutar (RUN) el programa en assembly para poner en la memoria el código máquina. Después cargue (LOAD) el programa a modificar y digite:

```
P%=PAGE
```

Esto pasa la dirección de comienzo del programa al programa de sustitución de variables. Luego ponga en PAGE un valor por encima de LOMEM, y cargue (LOAD) y ejecute (RUN) el programa de sustitución de variables. Puede volver al programa modificado mediante:

```
HIMEM=PAGE-1
PAGE=P%
```

Para utilizar la versión del programa para el Commodore 64, también necesitará poner en la memoria el código máquina, ya sea con el programa cargador de BASIC o bien ensamblando el código fuente. Si ensambla el código fuente, o carga el código máquina directamente como código máquina, tendrá que colocar (POKE) ceros en las direcciones 36864, 36865 y 36866. Esto equivale a renovar (NEW) la zona para programas alternativa.

Después de cargar el código máquina, SYS 49152 cambiará de una zona de programa a la otra. Si olvida dónde está, puede averiguarlo mediante PRINT PEEK(44), que dará 8 para la zona de programas normal y 144 para la zona de programas alternativa.

Una vez que tenga en el ordenador el código máquina, puede cargar (LOAD) el programa a modificar en la zona de programas normal y el programa para sustitución de variables en la zona de programas alternativa, ejecutando (RUN) luego el programa de sustitución de variables.

Conmutación y copia C64

```
 10 DIR=49152
 20 FOR LINEA=1000 TO 1180 STEP 10
 30 S=0
 40 FOR DIR=DIR TO DIR+7
 50 READ BYTE
 60 POKE DIR, BYTE
 70 S=S+BYTE
 80 NEXT DIR
 90 READ SUMACONTROL
100 IF S<> SUMACONTROL THEN PRINT"ERROR DE DATOS EN
    LINEA";LINEA:END
110 NEXT LINEA
120 POKE 36864,0:POKE36865,0:POKE36866,0
1000 DATA162,0,164,44,192,8,208,9,787
1010 DATA160,160,32,72,192,169,144,208,1137
1020 DATA7,160,144,32,72,192,169,8,784
1030 DATA162,1,134,43,133,44,134,45,696
1040 DATA133,46,160,0,177,45,170,200,931
1050 DATA177,45,224,0,208,240,201,0,1095
1060 DATA208,236,230,45,208,2,230,46,1205
1070 DATA136,16,247,165,45,133,47,133,922
1080 DATA49,165,46,133,48,133,50,96,720
1090 DATA134,51,132,52,134,55,132,56,746
1100 DATA202,136,134,53,132,54,96,160,967
1110 DATA0,177,251,164,255,145,251,160,1403
1120 DATA0,196,251,208,2,198,252,198,1305
1130 DATA251,165,253,197,251,208,234,165,1724
1140 DATA254,197,252,208,228,96,164,255,1654
1150 DATA177,251,160,0,145,251,230,251,1465
1160 DATA208,2,230,252,165,253,197,251,1558
1170 DATA208,236,165,254,197,252,208,230,1750
1180 DATA96,0,0,0,0,0,0,0,96
```

Copia BBC

```
 10 MODE 7
 20 HIMEM=HIMEM-&100
 30 CURR = &70
 40 LAST = &74
 50 DIFF = &78
 60 FOR PASS = 1 TO 2
 70 P%=HIMEM
 80[ OPT 1
 90 .UP          LDY #0
100 .UP1         LDA (CURR),Y
110              LDY DIFF
120              STA (CURR),Y
130              LDY #0
140              CPY CURR
150              BNE UP2
160              DEC CURR+1
170 .UP2         DEC CURR
180              LDA LAST
190              CMP CURR
200              BNE UP1
210              LDA LAST+1
220              CMP CURR+1
230              BNE UP1
240              RTS
250 .DOWN        LDY DIFF
260              LDA (CURR),Y
270              LDY #0
280              STA (CURR),Y
290              INC CURR
300              BNE DOWN1
310              INC CURR+1
320 .DOWN1       LDA LAST
330              CMP CURR
340              BNE DOWN
350              LDA LAST+1
360              CMP CURR+1
370              BNE DOWN
380              RTS
390]
400 NEXT PASS
410PRINT"UP,"DOWN
```

Sustitución de variables

Commodore 64

Introduzca estas modificaciones en el programa de p. 1180:

```
30005 INPUT"REEMPLAZAR POR";RS
30006 CURR=251
30007 LAST=253
30008 DI=255
30035 GOSUB 30500
30036 IF ERR THEN PRINT"NO ENCUENTRO EL
      FINAL DEL PROGRAMA":END
30060 IF SIGLINEA=0 THEN END
30065 CURRLINE TEXTPTR
```

Eliminar la línea 30085

```
30460 IF NOMBRE$=T$ THEN GOSUB 30600
30465 IF ERR THEN PRINT"NO HAY LUGAR
      EN LA LINEA";NUMLINEA:END
```

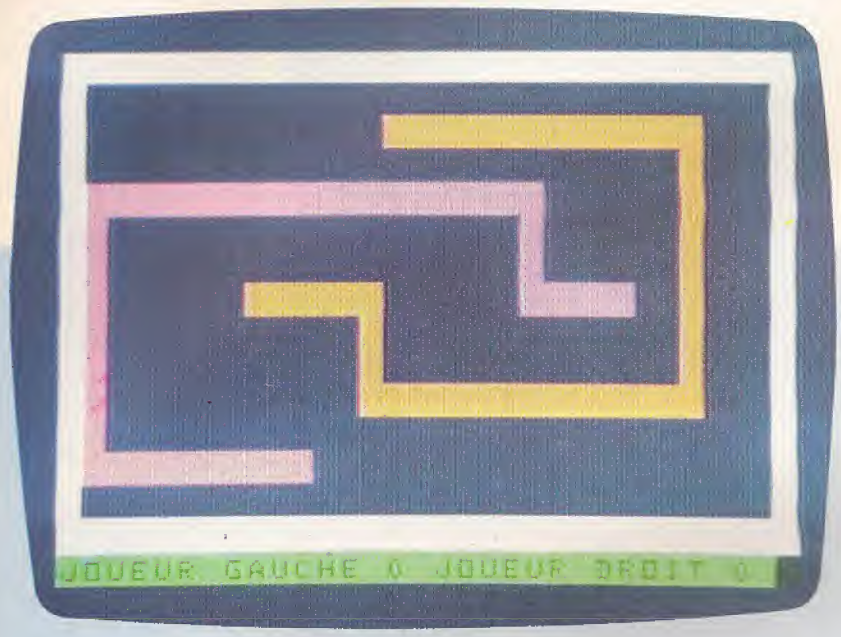
BBC Micro

Introduzca estas modificaciones en el programa de p. 1145:

```
30005 INPUT"Reemplazar por";
      SUSTITUCION$
30006 CURR = &70
30007 LAST = &74
30008 DIFF = &78
30055 GOSUB 30500
30056 IF Outofroom THEN PRINT"No encuentro
      el final del programa":PRINT"PAGE en
      lugar equivocado?":END
30060 Apuntadortexto=P%
30070 IF Numlinea > 32767 THEN END
30105 Bytelongitud=Apuntadortexto
30460 IF Nombre$=Objetivo$ THEN GOSUB
      30600
30465 IF Outofroom THEN PRINT"No hay lugar
      en la linea";Numlinea:END
```

Trazos

¿Una guerra de trincheras? No: este juego nos propone más bien una auténtica guerra de líneas. He aquí una versión para el microordenador Dragon



Dos jugadores se enfrentan para dividirse el espacio vital. Cada uno de ellos debe esforzarse, al irse desplazando, por no cortar jamás su trazado o el de su adversario y por no salirse del rectángulo dibujado en la pantalla. Utilice las palancas de mando o las siguientes teclas:

Jugador de la derecha: P, L, ; y .
Jugador de la izquierda: W, A, S y Z

```
10 REM*********
20 REM *TRAZOS*
30 REM *********
40 GOSUB 850
50 GOSUB 630
60 ON JK GOTO 150
70 FOR I=1 TO 50
80 NEXT I
90 D$=INKEY$
100 C1=(D$="L")-(D$";")+32*((D$="P")-
    (D$="."))
110 C2=(D$="A")-(D$="S")+32*((D$="W")
    -(D$="Z"))
120 IF C1<>0 THEN D1=C1
130 IF C2<>0 THEN D2=C2
140 GOTO 270
150 K0=JOYSTK (0)
160 K1=JOYSTK (1)
170 K2=JOYSTK (2)
180 K3=JOYSTK (3)
190 IF K0<26 AND K1>37 THEN S1=-1
200 IF K0>37 AND K1<26 THEN S1=1
210 IF K0<26 AND K1<26 THEN S1=-32
220 IF K0>37 AND K1>37 THEN S1=32
230 IF K2<26 AND K3>37 THEN S2=-1
240 IF K2>37 AND K3<26 THEN S2=1
250 IF K2<26 AND K3<26 THEN S2=-32
260 IF K2>37 AND K3>37 THEN S2=32
270 IF S1<>0 THEN D1=S1
280 IF S2<>0 THEN D2=S2
290 P1=P1+D1
300 IF PEEK (P1)<>128 THEN 360
310 POKE P1,J1
320 P2=P2+D2
330 IF PEEK (P2)<>128 THEN 410
340 POKE P2,J2
350 GOTO 60
360 F2=F2+1
370 GOSUB 590
380 IF F2=10 THEN 460
390 D$=INKEY$
400 GOTO 50
410 F1=F1+1
420 GOSUB 590
430 IF F1=10 THEN 500
440 D$=INKEY$
450 GOTO 50
460 CLS
470 PRINT@ 165, "GANA JUGADOR IZQUIERDA"
480 PRINT@ 202,F2;"A";F1
490 GOTO 530
500 CLS
510 PRINT@ 165, "GANA JUGADOR DERECHA"
520 PRINT@ 202,F1;"A";F2
530 R$=INKEY$
540 PRINT@266, "OTRA?"
550 R$=INKEY$
560 IF R$=" " THEN 540
570 IF R$<>"N" THEN RUN
580 END
590 FOR I=5 TO 255 STEP 5
600 SOUND I,1
610 NEXT I
620 RETURN
630 CLS 0
640 C=207
650 J1=239
660 J2=255
670 P1=1272
680 P2=1256
690 D1=-1
700 D2=1
710 FOR I=1024 TO 1055
720 POKE I,C
730 POKE I+448,C
740 NEXT I
750 FOR I=1 TO 13
760 POKE I*32+1024,C
770 POKE I*32+1055,C
780 NEXT I
790 PRINT@ 480, "JUGADOR IZQUIERDA";F2,
    "JUGADOR DERECHA";F1;
800 POKE P1,J1
810 POKE P2,J2
820 S1=0
830 S2=0
840 RETURN
850 CLS
860 PRINT@ 170, "PALANCAS MANDO?"
870 D$=INKEY$
880 IF D$=" " THEN 870
890 JK=-(D$="S")
900 RETURN
```

Visión periférica

Chris Stevens

He aquí un repaso a algunos conocidos periféricos y útiles indicaciones que el usuario debería tener en cuenta antes de adquirirlos

Hasta hace poco tiempo, el periférico más importante para el usuario de un ordenador personal era un módulo enchufable que contenía memoria extra. La memoria era cara y máquinas como el ZX-81 y el Vic-20 se diseñaron para mantener los costes al mínimo; de hecho, el ZX-81 no ampliado le ofrecía al usuario tan sólo setecientos y pico bytes con los que escribir programas. Las máquinas de hoy en día ya vienen equipadas con hasta 64 K de RAM (algunas, como el QL, ofrecen todavía más) y, por consiguiente, en la actualidad raramente se requieren "RAM *packs*". El usuario de hoy día tiene multitud de periféricos entre los que escoger: los modems permiten la comunicación entre usuarios que viven a cientos de kilómetros de distancia el uno del otro, los vehículos motorizados o los brazos-robot se pueden controlar utilizando una interface adecuada, y se pueden emplear sintetizadores de voz para entretenimiento o con fines educativos.

Si bien existen en el mercado muchos dispositivos periféricos diferentes, la mayoría sólo se fabrican para las máquinas más vendidas. Ésta es una consideración que se debe tener presente al decidir qué máquina adquirir; los usuarios de un Spectrum, un Commodore y un Acorn siempre tendrán más elecciones que quienes opten por un Oric o un Sord. La creación de un mercado de periféricos para las máquinas más modernas lleva su tiempo, si bien el recientemente introducido estándar MSX hará que las cosas resulten más fáciles al permitir que todas las máquinas que respondan a las especificaciones MSX utilicen los mismos accesorios.

Una consideración primordial para el comprador de periféricos es el tema de la compatibilidad: todo lo que se compre debe funcionar con cualquier otro dispositivo que se pueda llegar a adquirir en el futuro. El ejemplo clásico en este sentido es el del Spectrum. Muchos usuarios de esta máquina han comprado la Interface Uno y uno o dos microdrives, sólo para descubrir que algunos de los periféricos que ya poseían (e incluso parte de su software) no funcionan teniendo instalada la Interface Uno.

Sin embargo, de mantenerse la compatibilidad entre dispositivos, la elección de accesorios para su máquina puede incrementar en gran medida el atractivo que posee la informática. Periféricos adecuados pueden permitirle diseñar un sistema de ordenador que se adapte a sus propias exigencias particulares, y este sistema se puede ir modificando luego a medida que sus necesidades evolucionen.

Sistemas de almacenamiento

El dispositivo de almacenamiento más común para utilizar con un microordenador es la habitual grabadora de cintas de cassette de audio corrientes. Ésta ofrece la ventaja de ser sencilla de utilizar y relativamente económica, pero sus inconvenientes se hacen evidentes enseguida. Cargar programas ocupa mucho tiempo y es difícil llevar el registro exacto de qué es lo que hay en una cinta determinada. Las unidades de disco son más rápidas y más fiables, pero cuestan más. La mayoría de los ordenadores personales están limitados a un tipo de unidad de disco, y algunas de éstas (en particular los modelos de Commodore) son notablemente lentas en operación. La mayor parte de las máquinas personales continúan utilizando unidades de 5 1/4", pero ahora se están popularizando más las unidades de 3 o de 3 1/2". La unidad de disco Oric/Atmos, por ejemplo, emplea discos de 3" con una capacidad de 160 K por cada lado. El BBC Micro es sumamente flexible, ya que permite que se le conecten muchos sistemas de disco diferentes. El Torch Disk Pack convierte verdaderamente al BBC en un ordenador nuevo, con un microprocesador Z80 para complementar al 6502 del ordenador y 64 K de memoria adicionales. El Torch proporciona, asimismo, cuatro programas de "gestión" y viene con una versión del BASIC BBC que está diseñada para trabajar con el procesador Z80.

El Sinclair Spectrum, por el contrario, no posee medios para conectar unidades de disco estándares, si bien algunas firmas independientes han producido interfaces para disco especiales. Sinclair ha producido el sistema Interface Uno/Microdrive, que utiliza un bucle de cinta que puede almacenar alrededor de 85 K de datos. La cinta está completamente bajo el control del ordenador y cualquier dato individual se puede localizar en cuestión de unos 10 s. Esto representa un rendimiento que está a medio camino entre el de una grabadora de cassette y una unidad de disco, a un precio considerablemente inferior al del sistema de unidad de disco más económico. La Interface Uno ofrece la ventaja de proporcionar una interface RS232 (no estándar) y se puede utilizar para "conexión en red": el enlace en red de hasta 64 Spectrums o máquinas QL.

Un competidor del sistema Interface Uno es la Rotronics Wafadrive. Ésta también emplea bucles de cinta para retener datos, pero incluye interfaces RS232 y Centronics y un programa para tratamiento de textos incluido en su precio. Las cintas se suministran en tres tamaños diferentes (16, 64 y 128 K), obteniéndose la mayor velocidad operativa con la cinta de menor capacidad. En la fotografía vemos el Retronics Wafadrive, la unidad de disco Oric/Atmos, el Torch Disk Pack y la Interface Uno con microdrive de Sinclair

Dispositivos para gráficos

La producción de gráficos con un ordenador personal se simplifica considerablemente si se utiliza alguno de los muchos tipos diferentes de dispositivos para gráficos. Los más económicos son los lápices ópticos, que se pueden emplear para "dibujar" directamente en la pantalla de visualización mediante el empleo de una célula fotoeléctrica para detectar la posición de la punta del lápiz mientras toca la pantalla. Un desarrollo basado en este sistema es el Stack Light Rifle (véanse pp. 710-711).
A muchas personas les resulta difícil dibujar "a mano alzada" en una pantalla de visualización. Seguir líneas sobre una superficie plana es considerablemente más fácil y se fabrican muchos dispositivos que permiten que los usuarios hagan esto. Las tablillas para gráficos utilizan un lápiz especial que le transfiere al ordenador cualquier movimiento efectuado sobre la superficie de la tablilla; ello significa que se las puede utilizar para dibujar imágenes a mano alzada o para calcar imágenes impresas. Otros dispositivos de "dibujo" son los tiralíneas digitales, que hacen uso de resistencias variables situadas en un brazo mecánico para detectar la posición del punzón que llevan fijado en la punta del brazo. Aquí vemos la tablilla Grafpad de British Micro, el tiralíneas Robot Plotter y el lápiz óptico y Light Rifle de Stack.

Modems

El desarrollo de modems económicos para ordenadores personales permite a los usuarios comunicarse entre sí a través de la red telefónica. Para las máquinas equipadas con una interface RS232 estándar se han producido numerosos modems; con el software adecuado, éstos pueden acceder a la base de datos Prestel, que tiene muchas páginas dedicadas enteramente al usuario de ordenador personal. Asimismo, los modems se pueden utilizar para comunicarse con otros usuarios a través de "tablones de anuncios" (bases de datos frecuentemente mantenidas por entusiastas de los micros). Sin embargo, en este campo vuelve a plantearse la cuestión de la compatibilidad: distintos tablones de anuncios utilizan diferentes velocidades de transmisión, y un modem que puede utilizar el Prestel a menudo es inadecuado para comunicaciones con un tablón de anuncios. Ni el Spectrum ni el Commodore 64 poseen interface RS232 incorporada, y por consiguiente no pueden emplear modems estándares. Para el Spectrum, el modem más vendido es el Prism VTX5000. Mediante software en cinta dos Spectrums equipados con modems Prism pueden intercambiar programas o datos. Commodore suministra su propio modem para usar con el 64, y ha creado su propio sistema, Compunet, para conectar en red a los usuarios de Commodore 64.
Los usuarios de un modem deben estar siempre atentos al reloj, dado que los entusiastas pueden encontrarse enseguida con cuantiosas facturas telefónicas. Las tarifas anuales fijas para usuarios de Prestel y el Compunet son igualmente elevadas. La fotografía muestra el Prism VTX5000 y el modem de Commodore.

Sintetizadores de voz

Muchas máquinas personales populares pueden producir voz con la adición de una unidad para síntesis de voz. Las unidades existentes se pueden agrupar en dos categorías: una que se suministra con un vocabulario fijo de unas 100 palabras diferentes, y otra que utiliza *alófonos* (conjunto de diferentes sonidos y pausas a partir de los cuales se construyen las palabras).
La gama Currah de unidad de voz emplea el sistema de alófonos, y la firma produce módulos tanto para el Spectrum (Microspeech) como para el Commodore 64 (Speech 64). Algunos juegos para el Spectrum y el Commodore 64, en particular los producidos por Ultimate, poseen voz incorporada; ésta se produce automáticamente si se conecta una unidad Currah. La fotografía muestra el Speech 64 de Currah, y el Sweetalker de Cheetah para el Spectrum.

Chris Stevens

Dispositivos controlados por ordenador

Los ordenadores se pueden utilizar fácilmente para controlar dispositivos del mundo "real". La aplicación que se cita generalmente es el control de un sistema de calefacción central doméstico; ello es bastante factible, aunque casi no merece la pena, dado que tales sistemas suelen llevar incorporado un interruptor de reloj perfectamente eficaz. Mucho más interesantes son los diversos vehículos con ruedas. La Valiant Turtle es un vehículo que se parece algo a una tortuga y que puede producir los gráficos que utiliza el lenguaje LOGO. Este dispositivo se puede acoplar al Spectrum, al Commodore 64 y al BBC Micro, y emplea un haz infrarrojo para comunicarse con el ordenador. El BBC Buggy es un dispositivo de tipo similar.
En la fotografía vemos la Valiant Turtle y el BBC Buggy.

Impresoras-plotter

Para quien utilice un ordenador personal para el desarrollo de programas o para tratamiento de textos, la adquisición de una impresora resulta esencial. La mayoría son matriciales o de rueda margarita. Las primeras utilizan una cuadrícula de pequeños puntos para construir cada letra, permitiendo imprimir gráficos, y son de operación rápida, aunque su calidad de impresión es inferior a la de las de rueda margarita, que son básicamente máquinas de escribir controladas por ordenador. Un sistema alternativo es la pequeña impresora-plotter que se comercializa para los ordenadores Tandy, Atari, Commodore y Oric. Ésta utiliza un papel de unos 10 cm de ancho y está equipada con cuatro pequeños lápices de punta esférica que permiten escribir textos o producir gráficos multicolores. El texto se "dibuja" de la misma forma que los gráficos, y el dispositivo tiene programado un juego completo de caracteres. Otra alternativa es la que representa la impresora térmica Epson P40, que emplea una columna de elementos de calor para "quemar" los caracteres sobre papel especial. Esta es sumamente económica, a pesar de lo cual proporciona una calidad de impresión razonable y funciona con pilas recargables. También el papel que utiliza es bastante estrecho, pero puede producir una salida impresora de 80 columnas si se emplea la modalidad de impresión compacta. Aquí vemos la impresora-plotter (para Tandy/Radio Shack) y la Epson P40.

Palancas de mando

El primer periférico que suele adquirir el usuario de un ordenador personal es la palanca de mando. Muchos ordenadores están equipados con interfaces adecuadas, y algunas de las máquinas más modernas vienen ya con palancas de mando como estándar. Las palancas más comunes utilizan los conectores "D" de nueve patillas que fueron adoptados en primer lugar por los micros Commodore y Atari, y desde entonces lo han sido por numerosas empresas independientes. Las palancas BBC no son estándares, de modo que en este caso las opciones son más limitadas; en sus micros Plus/4 y 16, la firma Commodore ha ignorado su propio estándar, con lo cual ha obligado a los usuarios a emplear las nuevas palancas diseñadas por la misma.
Recientemente Sinclair ha comercializado la Interface Two (dos), interface para palanca de mando y puerta para cartucho de ROM para el Spectrum. Hasta ese momento no se había proporcionado para esta máquina ninguna interface "oficial" para palanca de mando, y el estándar de *facto* había sido la interface Kempston, cuyas especificaciones han sido adoptadas por muchas otras firmas. Lamentablemente son incompatibles y muchos juegos que son éxitos de ventas funcionarán bastante bien con la interface Kempston, pero no lo harán con la Interface Two, por lo cual ahora Kempston ha producido una que es compatible con el software escrito tanto para la Interface Two como con el antiguo formato Kempston. Una posible alternativa es comprar una interface "programable", que le permite al usuario ejecutar cualquier software, haya sido o no diseñado originalmente para utilizar con palanca de mando.
De las muchas palancas de mando que hay en el mercado, la más inusual es la nueva RAT de Cheetah (véase pp. 1070-1071). Ésta no posee ningún cable que la conecte al ordenador, sino que utiliza un haz infrarrojo para enviar y recibir señales. Por ahora, sólo está a la venta para el Spectrum. El sistema Amstrad es también un tanto inusual. El micro Amstrad posee un único conector para palanca de mando pero se le puede conectar a la primera una segunda palanca.
La fotografía muestra (en el sentido de las agujas del reloj): la palanca Amstrand, la RAT de Cheetah, la Kempston PRO 5000 y la interface Kempston para el ZX Spectrum.

Pantallas

La mayoría de los ordenadores personales se utilizan (al menos al principio) con un aparato de televisión normal como pantalla de visualización. Ello suele plantear problemas, dado que quizá otros miembros de la familia deseen mirar la televisión mientras el usuario del ordenador esté jugando al *Pacman*; de cualquier modo, la calidad de imagen con frecuencia es pobre. La solución es emplear una pantalla, que proporciona una imagen de mejor calidad. El usuario debe asegurarse de que compra el monitor adecuado, puesto que hay dos estándares principales: el RGB y el video compuesto (véase p. 509). Las pantallas de video compuesto se utilizan con micros Atari y Commodore, mientras que BBC, Oric/Atmos y Sinclair QL requieren el formato RGB, que es el que proporciona la mejor calidad de imagen. Algunos micros dependen del altavoz del televisor para producir sonido, de modo que los mismos requerirán pantallas con altavoz incorporado. Varios fabricantes de televisores producen aparatos equipados con interfaces para pantalla; éstos son ideales si el usuario requiere a la vez un televisor y una visualización de gran calidad. Aquí vemos el Microvitec Cub.

Creación de sonido

Ya nos es posible crear señales sonoras a partir del convertidor de digital a analógico diseñado anteriormente

El proyecto

El usuario puede controlar la salida a través de auriculares o bien a través de un estéreo, utilizando el par de conectores de salida que se hallan más cerca del potenciómetro de la caja D/A. Necesitará, asimismo, hacer algunos sencillos cables de conexión. Para auriculares: emplee el enchufe hembra del estéreo, consiga un conector en línea adecuado y suelde los dos extremos positivos a un cable conectado al conector rojo de la caja D/A. Para un estéreo: localice las conexiones audio-IN y tierra y luego haga el cable. Ahora siga estos pasos: conecte la caja buffer y el convertidor D/A y enchufe la caja buffer en la puerta para el usuario; enchufe los cables del amplificador de los auriculares o del estéreo en los conectores D/A; gire el potenciómentro del convertidor D/A totalmente hacia la izquierda y luego suminístrele potencia del transformador a la caja.

Ian McKinnell

Después de haber seguido estos pasos, podemos probar el sistema utilizando un corto programa en BASIC. En esencia, el sonido se genera eléctricamente proporcionándole a un micrófono un voltaje oscilante. Podemos generar una salida primaria de voltaje oscilante desde nuestro convertidor D/A cambiando el contenido del registro de datos de la puerta para el usuario de 0 a 255 y al revés en rápida sucesión. Entre el siguiente programa y ejecútelo. Gire el potenciómetro D/A en el sentido de las agujas del reloj hasta que se escuche sonido.

```
10 REM .... GENERADOR DE SONIDO BASIC CBM ....
20 RDD=56579:REGDAT=56577
30 POKERDD,255
35 N=1
40 POKE REGDAT,0:FORI=1TON:NEXT:POKE REGDAT,255:GOTO40
```

```
10 REM .... GENERADOR DE SONIDO BASIC BBC ....
20 RDD=&FE62:REGDAT=&FE60
30 ?RDD=255
35 N=1
40 ?REGDAT=0:FOR I=1TON:NEXT:? REGDAT=255:GOTO40
```

Observe que el programa en BASIC tiene una estructura repetitiva, concentrada toda en una única línea para conseguir la máxima velocidad. Hay un bucle de demora insertado entre los puntos en que el registro de datos se establece en 255 y su establecimiento en 0. El valor N de la línea 35 establece la longitud de esta demora. Pruebe de alterar el valor de N y ejecutar de nuevo el programa. Observará que la altura del tono oído baja mientras aumenta el valor de N.

```
      LDX #0          2
BUCLE1
      LDA TABLA,X     4
      STA PUERTA      4
      INX             2
      CMP #PASOS      2
      BNE BUCLE1      3
(2 si fracasa el bucle)
```

La mayor altura que se puede obtener mediante este programa en BASIC se producirá cuando el bucle de demora se elimine por completo. Incluso un bucle ejecutado una sola vez tiene un efecto audible sobre la altura de la nota escuchada.

Si ha experimentado con distintos valores de N en el programa en BASIC que hemos ofrecido, habrá notado que el cambio del valor de N en 1 tiene un efecto significativo sobre la altura de la nota. El BASIC no es lo suficientemente rápido como para permitirnos controlar con precisión la velocidad de oscilación. Para eso debemos utilizar el lenguaje máquina.

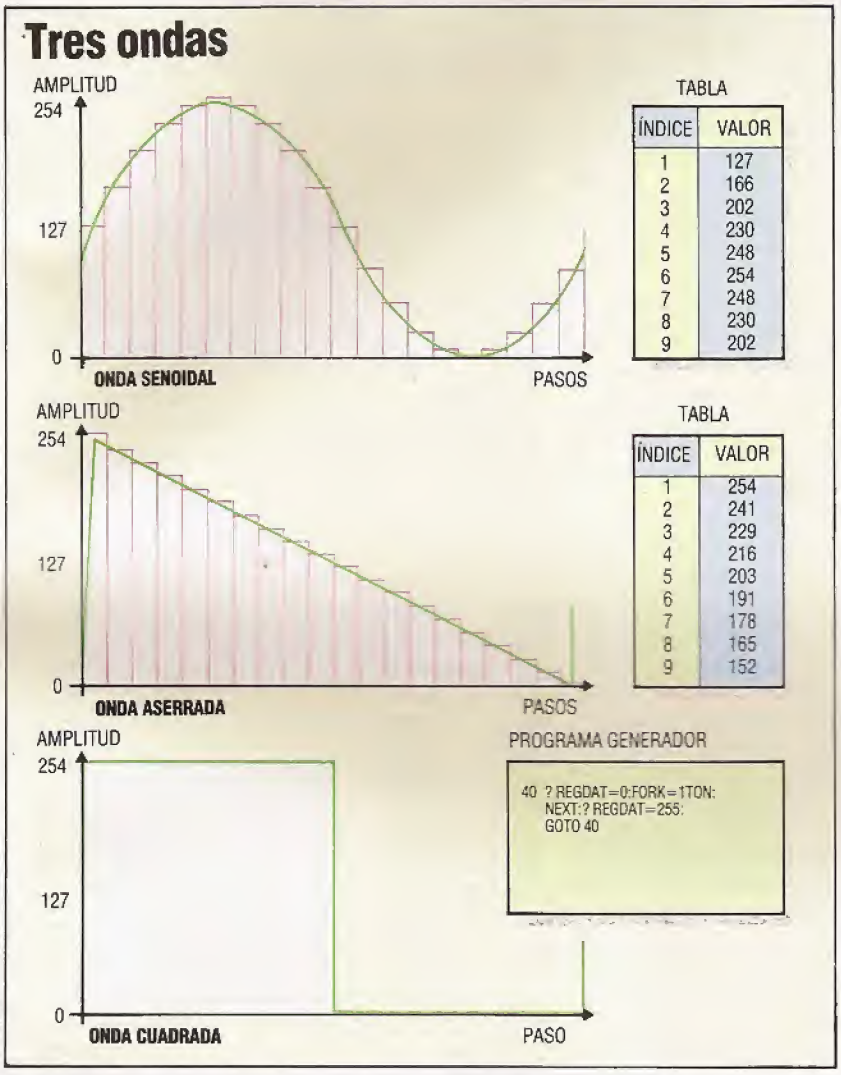
En el próximo capítulo de *Bricolaje* estudiaremos el difícil problema de controlar altura y volumen mediante código máquina. Aquí nos concentraremos en desarrollar un programa para producir distintas formas de onda. La forma de onda producida por el programa en BASIC utilizado anteriormente era una forma de onda cuadrada. No obstante, es posible producir otras formas de onda, que alteran la calidad del sonido oído. Podemos sintetizar digitalmente ondas senoidales y aserradas tomando un cierto número de muestras de la forma de onda y colocándolas en una tabla. El programa que se requiere en lenguaje máquina para colocar estas muestras una después de la otra en el registro de datos es esencialmente muy simple. Nuestra ilustración muestra estas tres formas de onda, acompañadas de las tablas para las formas de onda senoidal y aserrada. Se divide en pasos el ciclo de la forma de onda y se muestrean estos pasos; a la izquierda detallamos el bucle del programa que envía estas muestras a través de la puerta para el usuario.

Cuando se trata de producir sonido, el temporizado es esencial. A continuación de cada instrucción está la cantidad de ciclos de máquina requeridos para ejecutar esta instrucción. A partir de esta fórmula podemos calcular el número total de ciclos de máquina que lleva producir un ciclo completo de forma de onda: número de ciclos de máquina $= 2 + (4 + 4 + 2 + 2 + 3) \times$ pasos $- 1 = 1 + 15 \times$ pasos.

Si la onda se divide en 80 muestras, entonces el número de ciclos de máquina requeridos para producir un ciclo de forma de onda es 1 201.

Dado que en código 6502 cada ciclo de máquina toma alrededor de una millonésima de segundo, el número total de ciclos de forma de onda que se puede producir en un segundo (es decir, la frecuencia de la nota) viene dado por este cálculo: frecuencia = 1 000 000/1201 = 832 Hz. Puesto que do central es 512 Hz, la nota producida tendrá una altura de unas pocas notas más arriba de do central.

Se puede ver que el número de pasos de muestra



Liz Dixon

Creación de formas

Las formas de onda senoidal y aserrada se crean decidiendo primero cuántos pasos de muestra han de constituir un ciclo de la onda. Estas muestras de la amplitud de la onda se calculan luego y se almacenan en una tabla. Los valores de esta tabla se pueden, entonces, copiar en secuencia en el registro de datos de la puerta para el usuario y, por tanto, en el convertidor de digital a analógico, donde se convierten en niveles de voltaje. La ventaja de la tabla es que permite realizar de antemano los cálculos que tanto tiempo llevan; la verdadera generación de la forma de onda, por consiguiente, lleva poco tiempo, y ello hace posible una gama de frecuencias de varias octavas. Sin el uso de la tabla la gama se limitaría a dos octavas.

La onda cuadrada se puede generar mediante un programa en BASIC debido a que el proceso es muy sencillo. La lentitud de este lenguaje, sin embargo, limita considerablemente la gama de frecuencias por el cual hemos elegido dividir nuestra forma de onda tiene un efecto directo sobre la frecuencia de la nota final. Duplicando la cantidad de pasos de muestra dividiremos por la mitad la frecuencia de ésta. Obviamente, cuantas más muestras tomemos de la nota, más probabilidades tendremos de acercarnos a obtener la calidad de nota que estamos sintetizando, pero esto siempre se debe comparar con la frecuencia máxima final que se puede obtener para un número dado de pasos.

Es probable que un ciclo de forma de onda no tenga la longitud suficiente como para ser audible, por lo que debemos también incluir código para repetir la sección de código generadora de la forma de onda un número de veces establecido. La cantidad de repeticiones se puede determinar estableciendo un valor de contador y disminuyéndolo hasta cero. Para obtener una amplia gama de valores de contador, se ha utilizado un número de 16 bits almacenado en dos posiciones adyacentes. Además de este código, las interrupciones se desactivan al principio del programa mediante SE1 y se vuelven a activar mediante CL1 al final. Si se produjeran interrupciones durante la ejecución, ello restaría precisión al temporizado del programa. Sin embargo, no es posible desactivar todas las interrupciones; las interrupciones no enmascarables, si se producen durante la ejecución del programa, pueden causar algunos errores de temporizado.

Los datos de la forma de onda se deben colocar en la memoria dentro de una tabla, con cada tipo de forma de onda ocupando 80 posiciones consecutivas. En la versión Commodore, la tabla de la onda senoidal está ubicada en la memoria a partir de $C000; la tabla de la onda aserrada está en $C050 y la tabla de la onda cuadrada empieza en $C0A0. El programa en lenguaje máquina está diseñado para recurrir por defecto a cargar datos desde la tabla de onda senoidal utilizando el direccionamiento indexado, pero podemos pasar a otra tabla modificando el programa directamente con un POKE desde BASIC. La parte LDA de LDA SINE,X está en la posición $C103. La dirección de comienzo de la tabla de datos a cargar tiene su byte *lo* en la posición $C104 y su byte *hi* en la posición $C105. Para modificar la dirección de comienzo de los datos a cargar lo único que hay que hacer es cambiar el número retenido en $C104. Corrientemente, para una onda senoidal, esta posición contendrá 0; si deseamos cambiar a una onda aserrada, entonces deberemos cambiar el contenido de $C104 por 80. Cambiando el contenido de esta posición a 160 cambiaremos la forma de onda a una onda cuadrada.

La versión BBC también está diseñada de modo que las tablas empiezan al principio de una nueva página de la memoria. Por esta razón el byte *hi* de todas las direcciones de comienzo de las tablas es el mismo, y sólo hemos de modificar el byte *lo*. En un BBC Modelo B normalmente configurado en modalidad 7, HIMEM es &7C00. Bajando tres páginas el tope de la memoria destinamos un espacio más que suficiente para las tablas de referencia y el programa en lenguaje máquina.

Para el Commodore

```
                ;.....................................
                ;.....................................
                ;..                                 ..
                ;..        GENERADOR                ..
                ;..     DE SONIDO CBM 64            ..
                ;..                                 ..
                ;.....................................
                ;.....................................
                PUERTA   =56577          ;DIRECCION REGISTRO DATOS
                PASOS    =80             ;NUMERO DE PASOS EN ONDA
                *=$C000
                ;
                ;----PREPARAR ZONA TABLA DE DATOS----
                SENO     *=*+STEPS
                ASERR    *=*+STEPS
                CUADR    *=*+STEPS
                NUM      *=*+2
                CONT     *=*+2
                ;
                ;----PROGRAMA PRINCIPAL----
                ;
78              SEI
AD F0 C0        LDA NUM
8D F2 C0        STA CONT                ;ESTABLECER VALOR CONTADOR
AD F1 C0        LDA NUM+1
8D F3 C0        STA CONT+1
                ;
                LOOP2
A2 00           LDX #$00
                LOOP1
BD 00 C0        LDA SENO,X              ;TOMAR DATOS
8D 01 DD        STA PUERTA              ;PONER EN PUERTA USUARIO
E8              INX
E0 50           CPX #PASOS
D0 F5           BNE LOOP1               ;FIN DE UN CICLO
                ;
                ;----DECREMENTAR CONTADOR----
AD F2 C0        LDA CONT
38              SEC
E9 01           SBC #$01
8D F2 C0        STA CONT
AD F3 C0        LDA CONT+1
E9 00           SBC #$00
8D F3 C0        STA CONT+1
D0 E0           BNE LOOP2               ;SI HIBYTE>0
A9 00           LDA #$00
CD F2 C0        CMP CONT
D0 D9           BNE LOOP2               ;SI LOBYTE>0
58              CLI
60              RTS
```

El código máquina se puede entrar en su ordenador entrando el listado fuente proporcionado y ensamblándolo para crear un archivo hexa que se pueda cargar siempre que así se requiera. Las tablas se pueden generar ejecutando el siguiente programa:

```
 900 REM***PROGRAMA LLAMADA DE SONIDO CBM***
 910 :
 915 DN=8:REM PARA CASSETTE DN=1
 920 IF A=0 THEN=1:LOAD"5SOUND.HEX",DN,1
 999 :
1000 REM****PREPARAR VALORES DATOS****
1005 S=80          : REM NUMERO DE PASOS
1007 TB=12*4096   :REM COMIENZO DE ZONA DATOS
1008 :
1010 REM**ONDA SENOIDAL**
1020 FOR I=0 TO S-1
1030 Y=127*SIN(X)+127
1040 POKE TB+I,Y
1045 X=X+2/S
1050 NEXT I
1060 :
1065 REM****ONDA ASERRADA****
1070 Y=255:TB=TB+S
1080 FOR I=0 TO S-1
1090 POKE TB+I,Y
1100 Y=Y-255/S
1110 NEXT I
1120 :
1125 REM****ONDA CUADRADA****
1130 Y=255:TB=TB+S
1140 FOR I=0 TO S/2-1
1150 POKE TB+I,Y
1160 NEXT I
1165 Y=0
1170 FOR I=S/2 TO S-1
1180 POKE TB+I,Y
1190 NEXT I
1999 :
2000 REM****VISUALIZAR TABLAS DATOS****
2005 TB=12*4096
2010 FOR I=TB TO TB+3*S-1
2020 PRINTI,I-TB,PEEK(I)
2030 NEXT
```

Después de ejecutar este programa, digite NEW y después entre este programa muestra que ilustra cómo utilizar el código máquina, dando las direcciones SYS y POKE requeridas para interactuar con el código máquina desde BASIC. Este programa le pide al usuario que entre el tipo de onda requerido y luego produce un tono cada vez que se pulsa una tecla.

```
 10 REM **** SONIDO CBM 64 ****
 20 REM **** PROGRAMA MUESTRA ****
 30 :
 40 RDD=56579:POKE RDD,255: REM TODAS SALIDA
 65 CL=49392 :REM POSICION BYTE LO CONTADOR
 67 TL=49412 :REM DIGITAR POSITION BYTE LO
 70 SONIDO=49396 :REM DIRECCION DE COMIENZO PROGRAMA
 75 REM ** ESTABLECER VALOR CONTADOR **
 80 NUM=80:NHI=INT(NUM/256):NLO=NUM-256*NHI
 82 POKE CL,NLO:POKE CL+1,NHI
 83 :
 85 PRINTCHR$(147):REM BORRAR PANTALLA
 86 INPUT"TIPO DE ONDA (0)SENOIDAL (1)ASERRADA (2)CUADRADA";WT
 87 POKE TL, WT*S
 88 PRINT:PRINT"PULSE CUALQUIER TECLA (RUN/STOP PARA TERMINAR)"
 90 GETA$:IF A$=""THEN90:REM ESPERAR UNA TECLA
100 SYS SOUND:REM LLAMAR CODIGO MAQUINA
110 IF A$="X" THEN 85
120 GOTO 90
```

Si no posee ensamblador o no entiende el lenguaje assembly aun así puede utilizar el programa en código máquina digitando este cargador en BASIC y ejecutándolo. En este caso, puede omitir la línea 920 del programa que prepara la tabla.

```
 10 REM **** CARGADOR BASIC PARA SONIDO CBM ****
 20 REM **** CODIGO MAQUINA ****
 30 FOR I=49396 TO 49449
 40 READ A:POKE I,A
 50 CC=CC+A
 60 NEXT I
 70 READ CS:IF CC<>CS THEN PRINT"ERROR SUMA CONTROL":END
100 DATA120,173,240,192,141,242,192
110 DATA173,241,192,141,243,192,162,0
120 DATA189,0,192,141,1,221,232,224,80
130 DATA208,245,173,242,192,56,233,1
140 DATA141,242,192,173,243,192,233,0
150 DATA141,243,192,208,224,169,0,205
160 DATA242,192,208,217,88,96
170 DATA9115:REM*SUMA DE CONTROL*
```

Para el BBC

En el BBC, el proceso de combinar BASIC y código máquina es más fácil que en el caso del C64.

```
  5 REM **** PROGRAMA DE SONIDO BBC ****
  8 MODE 7
 10 HIME=HIMEM-&0301
 20 MC%=HIMEM+1
 30 RDD=&FE62:?RDD=255:REM TODAS SALIDA
 40 puerta=&FE60:REM REG DATOS FUERA USUARIO
 50 pasos=80     :REM NUMERO DE PASOS POR ONDA
 60 comienzo_tabla=MC%
 70 PROCpreparar_tablas
 80 PROCcodigo_maquina
 90 PROCprograma_muestras
999 END
1000 DEF PROCcodigo_maquina
1010 :
1020 FOR opt%=1 TO 3 STEP 3
1030 P%=MC%
1060 seno=P%:          P%=P%+pasos
1070 aserr=P%:         P%=P%+pasos
1080 cuadr=P%:         P%=P%+pasos
1090 num=P%:           P%=P%+2
1100 cont=P%:          P%=P%+2
1110 [
1120 OPT opt%
1130 \**** AQUI COMIENZA EL PROGRAMA PRINCIPAL ****
1150 .sonido
1160            SEI
1170            LDA num
1180            STA cont
1190            LDA num+1
1200            STA cont+1
1220 .bucle2
1230            LDX #&00
1240 .bucle1
1250            LDA seno,X
1260            STA puerta
1270            INX
1280            CPX #pasos
1290            BNE bucle1
1310 \**** DECREMENTAR CONTADOR ****
1320 \
1330            LDA cont
1340            SEC
1350            SBC #&01
1360            STA cont
1370            LDA cont+1
1380            SBC #&00
1390            STA cont+1
1400            BNE bucle2
1410            LDA #&00
1420            CMP cont
1430            BNE bucle2
1440            CLI
1450            RTS
1455 ]
1460 NEXT opt%
1480 ENDPROC
2000 DEF PROCpreparar_tablas
2020 REM **** ONDA SENOIDAL ****
2025 x=0
2030 FOR I=0 TO pasos-1
2040 y=127*SIN(x)+127
2050 ?(comienzo_tabla+I)=y
2060 x=x+2*PI/pasos
2070 NEXT I
2090 REM **** ONDA ASERRADA ****
2100 y=255:comienzo_tabla=comienzo_tabla+pasos
2110 FOR I+0 TO pasos-1
2120 ?(comienzo_tabla+I)+y
2130 y=y-255/pasos
2140 NEXT I
2160 REM **** ONDA CUADRADA ****
2170 y=255:comienzo_tabla=comienzo_tabla+pasos
2180 FOR I=0 TO pasos/2-1
2190 ?(comienzo_tabla+1)=y
2200 NEXT I
2220 y=0
2230 FOR I=pasos/2 TO pasos-1
2240 ?(comienzo_tabla+I)=y
2250 NEXT I
2270 REM **** VISUALIZAR TABLAS DE DATOS ****
2280 comienzo_tabla=MC%
2290 FOR I=comienzo_tabla TO comienzo_tabla+3*pasos-1
2300 PRINT I, ?(I-comienzo_tabla),? I
2310 NEXT I
2330 ENDPROC
3000 DEF PROCprograma_muestra
3020 cont=MC%+3*pasos:REM POSICION BYTE LO CONTADOR
3030 tipo=bucle1+1:REM POSICION BYTE LO TIPO
3040 valor_contador=80
3050 hi_contador=valor_contador DIV 256
3060 lo_contador=valor_contador MOD 256
3070 ?cont=lo_contador
3080 cont?1=hi_contador
3090 CLS
3100 INPUT"TIPO DE ONDA (0) SENOIDAL (1) ASERRADA (2) CUADRADA";onda
3110 ?tipo=onda*pasos
3120 REPEAT
3125 PRINT"PULSE CUALQUIER TECLA (X PARA SALIR)"
3130 A$=GET$
3140 CALL sonido
3150 UNTIL A$="X"
3160 GOTO 3090
```

Manejando cifras

En este capítulo de nuestro curso analizaremos las facilidades que ofrece el LOGO para trabajar con números

Casi todas las implementaciones del LOGO soportan tanto aritmética de enteros como de reales (decimales), utilizando los operadores infijos + − * /. Estos operadores se denominan infijos porque se escriben entre los números con los cuales trabajan; por ejemplo, 3+4. Algunas versiones de LOGO incluyen también aritmética de "prefijos", según la cual nuestro ejemplo se escribiría SUM 3 4. Una ventaja de esta notación es que es coherente con la forma en que se escriben las otras operaciones e instrucciones del LOGO.

El LOGO MIT sólo soporta aritmética de infijos, pero es sencillo programar formas con prefijos si así se requiere. Defina SUMA y PRODUCTO y pruébelos:

```
TO SUMA :A :B
  OUTPUT :A + :B
END

TO PRODUCTO :A :B
  OUTPUT :A * :B
END
```

La "precedencia" de operaciones (el orden por el cual se llevan a cabo) sigue las reglas matemáticas habituales. Todo lo que esté entre paréntesis se realiza primero, seguido de multiplicaciones y divisiones, y finalmente sumas y restas:

```
PRINT (3 + 4) * 5
PRINT 3 + 4 * 5
```

Ahora pruebe las formas de prefijo:

```
PRINT PRODUCTO 5 SUMA 3 4
PRINT SUMA 3 PRODUCTO 4 5
```

Esto demuestra otra ventaja de las formas de prefijo: no existe necesidad alguna de reglas de precedencia y la línea se evalúa de la misma manera que cualquier otra línea de instrucciones del LOGO.

La habitual operación de la división (/) da el resultado como un número real. Otras dos operaciones, QUOTIENT (cociente) y REMAINDER (resto), suelen ser útiles para trabajar con enteros.

```
QUOTIENT 47 5 es 9
REMAINDER 47 5 es 2
```

Un método estándar para convertir un número en base 10 a binario es ir dividiendo el número por dos hasta que el resultado sea cero. El número binario resultante se halla escribiendo los restos de cada etapa por orden inverso. Por ejemplo, para convertir 12 a binario:

```
12/2 = 6;resto = 0
 6/2 = 3;resto = 0
 3/2 = 1;resto = 1
 1/2 = 0;resto = 1
```

De este modo, leyendo los restos hacia arriba, hallamos que el decimal 12 es 1100 en binario.

Utilizando QUOTIENT y REMAINDER podemos implementar esta técnica en LOGO fácilmente. Colocando la sentencia de impresión *después* de la llamada recursiva obtenemos los restos impresos en el orden correcto (inverso).

```
TO BIN :X
  IF :X=0 THEN STOP
  BIN QUOTIENT :X 2
  PRINT1 REMAINDER :X 2
END
```

Existen dos operaciones para redondear números: INTEGER y ROUND. INTEGER produce la parte entera de un número, simplemente ignorando cualquier cifra que haya después del punto decimal, y ROUND redondea un número aumentándolo o disminuyéndolo hasta el número entero más cercano.

Los siguientes procedimientos calculan el interés compuesto sobre una inversión a un porcentaje dado de interés. En IMPRESION.EXCELENTE, INTEGER se utiliza para obtener las pesetas, y ROUND, para redondear los céntimos al número entero más cercano.

```
TO COMPUESTO :PRINCIPAL :PRCNT :AÑOS
  IF :AÑOS=0 THEN IMPRESION.EXCELENTE
  :PRINCIPAL STOP
  COMPUESTO :PRINCIPAL*(1+ :PRCNT/100)
    :PRCNT :AÑOS − 1
END

TO IMPRESION.EXCELENTE :DINERO
  MAKE"PESETAS INTEGER :DINERO
  MAKE"CENTIMOS ROUND (:DINERO −
    :PESETAS)*100
  (PRINT :PESETAS"PESETAS :CENTIMOS
    "CENTIMOS)
END
```

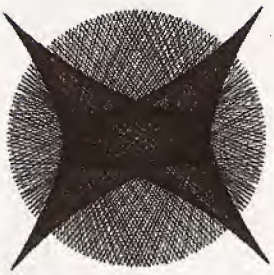
Hora de probar

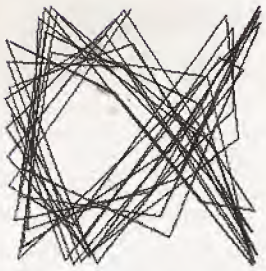
Anteriormente ya hemos utilizado =, < y > como operadores lógicos en comparaciones en numerosos procedimientos. Se pueden emplear las operaciones lógicas ALLOF, ANYOF y NOT para combinar otras condiciones. ALLOF es verdadera si sus dos entradas son verdaderas, ANYOF es verdadera si alguna de sus entradas es verdadera, y NOT es verdadera si su entrada es falsa. De modo que tenemos:

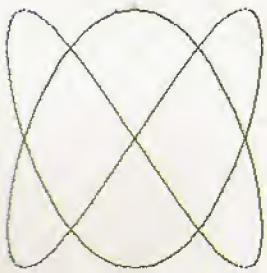
```
IF ANYOF :X > 0 :X=0 THEN PRINT"POSITIVO
IF NOT :X < 0 THEN PRINT "POSITIVO
IF ALLOF :X > 0 :X <100 THEN PRINT
  [ENTRE 0 Y 100]
```

La operación NUMBER? produce TRUE (verdadero) si la entrada es un número, de lo contrario devuelve FALSE (falso). La utilizamos en el procedimiento PRIMO?, que produce TRUE si su entrada es un número primo, o, de lo contrario, FALSE. Empieza por verificar que la entrada sea realmente un núme-

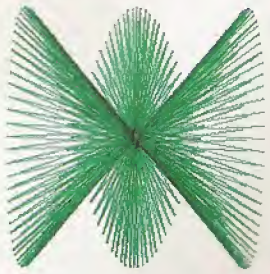
FIGURAS LISSAJOUS

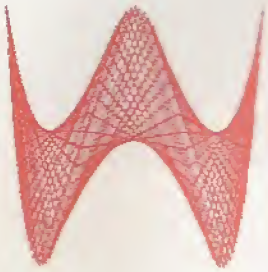






FIGURAS LISSAJOUS





Caminando sobre la línea

El teorema del "recorrido del borracho" postula que al cabo de N pasos en direcciones totalmente aleatorias, la probabilidad de que la distancia del borracho respecto al punto de origen sea inferior a la raíz cuadrada de N pasos es mayor de 0,5. Esta es una predicción estadística basada en una gran cantidad de pasos; el Logo le ofrece la posibilidad de probarla:

```
TO BORRACHO :NUM PASOS:
  :PASO
CS REPEAT :NUM PASOS [RT
  (RANDOM 361) FD
  :PASO]
END
```

ro y que sea mayor que dos. PRUEBA.PRIMO verifica entonces si hay algún entero entre la raíz cuadrada del número y dos que sea divisible por el mismo, sin que quede ningún resto.

```
TO PRIMO? :NUM
  IF NOT NUMBER? :NUM THEN PRINT [NO ES
    UN NUMERO] STOP
  IF :NUM < 2 THEN OUTPUT"FALSE
  OUTPUT PRUEBA.PRIMO :NUM INTEGER SQRT
    :NUM
END

TO PRUEBA.PRIMO :NUM :HECHO
  IF :HECHO=1 THEN OUTPUT"TRUE
  IF (REMAINDER :NUM :HECHO)=0 THEN OUTPUT
    "FALSE
  OUTPUT PRUEBA.PRIMO :NUM :HECHO - 1
END
```

Números aleatorios

RANDOM n produce un entero al azar entre 0 y n−1. El procedimiento BORRACHA hace que la tortuga trastabillee a través de la pantalla, girando en un ángulo al azar a cada paso. La entrada A da el tamaño máximo del giro que se puede describir en cualquier momento. Si ejecuta este procedimiento comprobará que la tortuga gira en círculos imprecisos, moviéndose hacia la izquierda o la derecha según el valor que se le haya asignado a A.

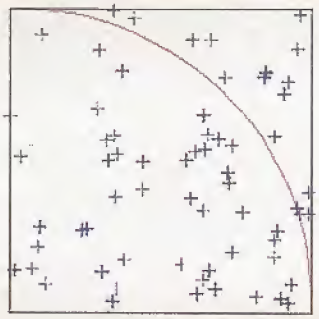
```
TO BORRACHA :A
  FORWARD 1
  RIGHT (-:A/2+RANDOM :A)
  BORRACHA :A
END
```

El llamado *método Monte Carlo* es una técnica para resolver problemas matemáticos mediante el empleo de números aleatorios.

RECORRIDO DEL BORRACHO

Steve Malone

"PI" LLEGA A MONTE CARLO



En nuestra demostración hallaremos una aproximación a *pi* empleando este método. La ilustración muestra un cuarto de círculo dibujado dentro de un cuadrado. La superficie del cuadrado es de 100 × 100 unidades cuadradas, y la superficie del cuarto de círculo es (1 ÷ 4) × *pi* × 100 × 100 unidades cuadradas. La razón de las superficies círculo ÷ cuadrado es igual a *pi* ÷ 4. Ahora clave un alfiler al azar sobre el cuadrado 1 000 veces y cuente cuántas veces el alfiler ha caído dentro del cuarto de círculo; a este número llámelo IN. El valor de IN/1000 debe ser aproximadamente el mismo que el resultado de: círculo ÷ cuadrado, es decir, *pi* ÷ 4. De modo que si hacemos el experimento, multiplicamos IN por cuatro y dividimos por 1 000, el resultado debería ser una aproximación a *pi*. Eso es precisamente lo que hacen los siguientes procedimientos:

```
TO MC
  DRAW
  PU
  MAKE"IN 0
  MC1 1000 100 100
  (PRINT [EL VALOR DE PI ES]0.004*(:IN))
END

TO MC1 :NUM :XNUM :YNUM
  IF :NUM = 0 THEN STOP
  PUNTO.ALEATORIO :XNUM :YNUM
  IF DENTRO? THEN MAKE "IN :IN + 1
  MC1 :NUM-1 :XNUM :YNUM
END
```

El procedimiento MC simplemente establece las condiciones, llama a MC1 e imprime los resultados. MC1 realiza la mayor parte del trabajo, llama a PUNTO.ALEATORIO para posicionar la tortuga y luego incrementa IN si el punto está dentro del círculo. Esto continúa hasta que el procedimiento se haya llevado a cabo la cantidad correcta de veces.

```
TO PUNTO.ALEATORIO :XNUM :YNUM
  SETXY RANDOM :XNUM RANDOM :YNUM
END

TO DENTRO?
  IF (XCOR*XCOR+YCOR*YCOR) < 10000
    THEN OUTPUT "TRUE
  OUTPUT "FALSE
END
```

PUNTO.ALEATORIO coloca a la tortuga dentro del cuadrado en una posición al azar, mientras que DENTRO? verifica si la tortuga está dentro del círculo. Ejecutar esto llevará bastante tiempo, pero finalmente se obtendrá un valor para *pi* de 3,15999.

Las curvas Lissajous son una interesante familia de curvas en las cuales la coordenada *x* de cada

punto está determinada por la función seno y la coordenada *y* por la función coseno:

```
TO LJ :COEF1 :COEF2 :PASO
  DRAW PU HT
  POS :COEF1 :COEF2 0 PD
  LJ1 :COEF1 :COEF2 0 :PASO
END

TO POS :COEF1 :COEF2 :ANGULO
  MAKE "X 100*SIN (:COEF1* :ANGULO)
  MAKE "Y 100*COS (:COEF2* :ANGULO)
  SETXY :X :Y
END

TO LJ1 :COEF1 :COEF2 :ANGULO :PASO
  POS :COEF1 :COEF2 :ANGULO
  LJ1 :COEF1 :COEF2 (:ANGULO + :PASO) :PASO
END
```

Complementos al LOGO

Las versiones LCSI incluyen aritmética de prefijo. El LOGO Atari posee SUM y PRODUCT; el LOGO Spectrum posee también DIV, y el LOGO Apple tiene QUOTIENT, que corresponden ambas a QUOTIENT del LOGO MIT.
Se utiliza INT en lugar de INTEGER.
En vez de NUMBER? se emplea NUMBERP.
Los operadores lógicos poseen los nombres más usuales de AND, OR y NOT.
IF tiene una sintaxis distinta: en lugar de PRINT1 se utiliza IF :X = 0 PRINT "TIPO CERO.
En lugar de SETXY se utiliza SETPOS (seguido de una lista). Utilice CS en lugar de DRAW.

Ejercicios de LOGO

1. Escriba un procedimiento para producir la enésima potencia de un número, de modo que POTENCIA 4 2 produzca 16.
2. Escriba un conjunto de procedimientos para convertir un número decimal a hexadecimal.
3. Escriba un procedimiento PAR? que produzca TRUE si un número es par y FALSE si no lo es.
4. Utilice el método Monte Carlo para hallar la superficie existente bajo la curva $y=x^2$ entre $x=0$ y $x=10$.

Respuestas a los ejercicios

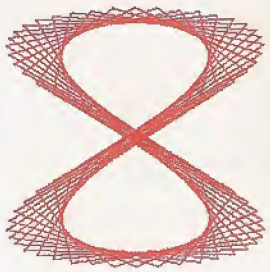
1. Juego de conversión para utilizar control por teclado:
Cambie ESTABLECER.DIABLOS MIRAR,VERIFICAR. Elimine ENCPAL. Agregue MOVER y TECLALEIDA.

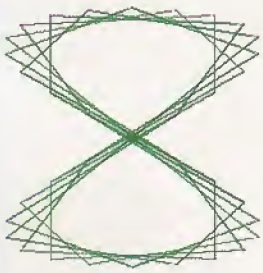
```
  TO ESTABLECER.DIABLOS
    WHEN OVER :OVEJA1 :CERCA [SETSP 0]
    WHEN OVER :OVEJA2 :CERCA [SETSP 0]
    WHEN TOUCHING :OVEJA1 :OVEJA2 [SETSP 0]
    WHEN TOUCHING :PERRO :OVEJA1 [SETSP 0]
    WHEN TOUCHING :PERRO :OVEJA2 [SETSP 0]
  END
  TO MIRAR
    MOVER TECLALEIDA
    IF :VELOCIDAD = 0 [VERIFICAR]
    MIRAR
  END
  TO VERIFICAR
    IF COND OVER :OVEJA1 :CERCA [ASK
      :OVEJA1 [BK 10 RT 90]]
    IF COND OVER :OVEJA2 :CERCA [ASK
      :OVEJA2 [BK 10 RT 90]]
    IF COND TOUCHING :OVEJA1 :OVEJA2
[CHOQUE]
    IF COND TOUCHING :PERRO :OVEJA1 [ASK
      :OVEJA1 [RT 90]]
    IF COND TOUCHING :PERRO :OVEJA2 [ASK
      :OVEJA2 [RT 90]]
    ESTABLECER.VELOCIDADES
  END
  TO MOVER :COM
    IF :COM = "W [ASK :PERRO [SETH 0]]
    IF :COM = "S [ASK :PERRO [SETH 90]]
    IF :COM = "Z [ASK :PERRO [SETH 180]]
    IF :COM = "A [ASK :PERRO [SETH 270]]
    IF :COM = "Q [ASK :TORTUGA [DIBUJAR.
      CAJA]]
  END
  TO TECLALEIDA
    IF KEYP [OUTPUT RC]
    OUTPUT"
  END
```

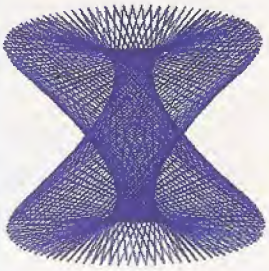
2. El juego de los meteoritos: defina forma 1 como un meteorito y forma 2 como la nave espacial.

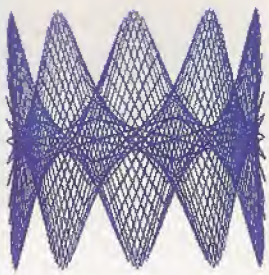
```
  TO JUGAR
    CS FS
    EST 0 1 [– 100 80] 180 199
    EST 1 1 [0 80] 180 199
    EST 2 1 [100 90] 180 199
    EST 3 2 [0 – 80] 90 50
    ESTABLECER.DIABLOS
    MOVER.AZAR 0
  END
  TO EST :NUM :FORMA :POS :CABEZA :VEL
    TELL :NUM SETSH :FORMA
    PU SETPOS :POS
    SETH :CABEZA ST SETSP :VEL
  END
  TO ESTABLECER. DIABLOS
    WHEN TOUCHING 0 3 [BANG]
    WHEN TOUCHING 1 3 [BANG]
    WHEN TOUCHING 2 3 [BANG]
    WHEN 15 [ENCPAL]
  END
  TO BANG
    TELL [0 1 2 3]
    SETSP 0 SS
    PRINT " PRINT "
    PRINT "SALPICADO
  END
  TO ENCPAL
    IF (JOY 1) < 0 [STOP]
    ASK 3 [SETH 45* JOY 1]
  END
  TO MOVER.AZAR :NUM
    IF SPEED = 0 [(PRINT "MARCADOR :NUM)
      STOP]
    ASK RANDOM 3 [SETH 145 + RANDOM 70]
    MOVER.AZAR :NUM+1
  END
```

FIGURAS LISSAJOUS









Saber diseñar el programa

Hasta ahora nos hemos limitado a construir rutinas sencillas. Es el momento de abordar tareas más ambiciosas, como la estructuración de programas en assembly

El diseño importa
Resultan difíciles de observar las reglas de una buena estructura cuando se programa en lenguaje máquina. Pero no es difícil crear un programa según las reglas de un buen diseño, además de ser un instrumento excelente de claridad y de ahorro de tiempo en la depuración

Mucho llevamos escrito sobre las ventajas de un adecuado diseño de los programas, la construcción por módulos y la programación estructurada en el contexto de los lenguajes de alto nivel. Pero tanto las dificultades como las ventajas se encuentran agrandadas en los de bajo nivel. No existen en assembly estructuras apropiadas de control como las hay en BASIC (p. ej., WHILE...WEND, IF...THEN...ELSE) que le proporcionen algún tipo de estructura al código. Tampoco se dispone de notaciones adecuadas, diversos tipos de variables, y ni siquiera se puede pedir a un programa en assembly que no sea seis o diez veces mayor, en número de instrucciones, que el mismo programa escrito en un lenguaje de alto nivel. Y encima, es mucho más fácil que se cometan errores de consecuencias desastrosas, tales como borrar todos los datos introducidos en un disco por culpa de un mínimo error existente en un solo byte. Con el fin de que la programación en assembly resulte menos aventurada, vamos a exponer un método para abordarla.

No hay nada absolutamente nuevo en la programación estructurada o ingeniería de software: un programador experimentado sabe sencillamente que tener el problema totalmente especificado y la claridad en el análisis constituyen la base ineludible de un estilo de programación con éxito. Nada más ilustrativo para probar esto que tratar de descubrir errores en un programa escrito en lenguaje máquina por usted de una manera poco estructurada e indocumentada y que dejó en algún sitio hace algunos meses. Un buen diseño y unos adecuados métodos de trabajo garantizan por sí solos un buen programa.

Fases en el diseño del programa

- *Especificar el problema.* En esta fase, el programador debe cuidar sobre todo de especificar cuáles son las entradas y cuáles las salidas. A menudo los dispositivos periféricos son controlados directamente (en especial, el teclado y la pantalla), por lo que se han de tener en cuenta las señales utilizadas. Puede que también existan limitaciones de tiempo. Puede que usted carezca de rutinas apropiadas para convertir una cadena de bytes que entra o sale en la forma en que el programa lee los datos (p. ej., la conversión de una cadena de caracteres ASCII en un número decimal en formato binario). Es, pues, importante especificar no sólo el formato en que surge el dato sino también el que será exigido por el resto del programa.

- *Diseño del programa.* Se trata ahora de idear los procesos por los que la entrada especificada alcanza la salida o resultado deseado. Tales procesos han de agruparse por módulos en la medida de lo posible, autosuficientes desde un punto de vista lógico, junto con los datos que cada proceso necesita. Hay dos técnicas principales para "cortar" un programa en módulos: 1) La técnica *bottom-up* (de abajo arriba), en la que se hace una recolección de módulos que pueden resultar útiles en el contexto del programa para después ajustarlos; 2) La técnica del *top-down* (de arriba abajo), en la que se va descomponiendo sucesivamente el programa en unidades cada vez más pequeñas, interesándonos antes en la función a desempeñar que en cómo lograrla, hasta el punto en que el proceso no puede seguir más adelante. Sólo en ese momento se comienza a pensar en el modo de codificar cada módulo.

El diseño *bottom-up* ofrece la ventaja de emplear módulos de librería, fáciles de conjuntar y cuyo resultado suele ser un uso más eficaz de la memoria. La desventaja está en que el programa en su totalidad puede resultar difícil de depurar y comprobar, así como puede que no sea tan comprensible. El diseño *top-down* proporciona programas mejor estructurados, y cada fase del proceso puede ser comprobada por separado por medio de "colillas" (*stubs*), breves rutinas que reemplazan los módulos todavía por escribir limitándose a aceptar entradas y proporcionando una salida correcta sin realizar proceso alguno. La desventaja se halla en que los programas probablemente ocuparán mucha más memoria y difícilmente las rutinas creadas podrán tener un uso inmediato en otro lugar.

Dentro de cada módulo se especificarán los datos que se necesitan, sus estructuras y algoritmos. En este nivel resulta útil un diagrama de flujo para representar los algoritmos, pero hay quienes prefieren, por más sencillo, trabajar libremente con un lenguaje de alto nivel llamado seudocódigo. La base de este seudocódigo suele ser el PASCAL, pero no hay razón alguna para prescindir del BASIC. Éste nos permite diseñar algoritmos y datos de una manera que nos es familiar, y relega el trabajo de bajo nivel a las tareas más sencillas de traducir a assembly los algoritmos en seudocódigo. Lo cual es mucho más fácil que intentar diseñar y codificar en assembly al mismo tiempo.

- *Codificación.* Si las rutinas fueron correctamente diseñadas, ésta será sin duda la fase más sencilla y

breve de todas. Para traducir un algoritmo en alto nivel a código en bajo nivel es esencial que se trasladen al bajo nivel las estructuras de control empleadas en alto nivel, desechando la tentación de emplear BRA y JMP indiscriminadamente. Recuerde que el tiempo que ahorra escribiendo código no estructurado se va a derrochar en una siguiente fase de depuración con sus frustrantes procesos de "ensayo y error".

En el diagrama de esta página proporcionamos algunos ejemplos de cómo pueden codificarse las más comunes estructuras de control, suponiendo, para mayor simplicidad, que los ítems de datos empleados son de ocho bits.

El problema de una codificación de este tipo de las estructuras de control es que el programa puede alargarse más de lo previsto. Esto es irrelevante si no se tiene limitación de memoria; una codificación más breve no siempre significa menor tiempo de ejecución, sino un desarrollo más lento y mucho tiempo invertido en la depuración. Pero si el espacio está limitado, es mejor escribir en una forma estructurada espaciosa e intercalar una fase de optimización en la que la codificación operativa puede abreviarse teniendo en cuenta situaciones particulares y respetando en lo posible la estructura esencial.

- *Depuración.* En esta fase se probará cada módulo por separado (empleando *stubs* donde sea necesario) para asegurarse que proporciona las salidas adecuadas a las entradas válidas. La depuración de programas en assembly dista mucho de parecerse a la del BASIC. Para poder observar lo que está sucediendo, es necesario inspeccionar el contenido de los registros y de las posiciones de memoria empleadas por el programa, y cambiarlo si llega el caso. Es casi imposible depurar un programa assembly sin la ayuda de una utilidad que permita poner y sacar puntos de ruptura (*breakpoints*). Estos puntos permiten ejecutar un programa hasta dicha ruptura, volcar los registros e inspeccionar y cambiar los contenidos de la memoria.

- *Comprobación.* Una vez comprobado cada módulo, hay que acoplar el programa entero y probarlo con datos apropiados. Esto es todavía más fácil cuando se sabe que cada una de las partes funciona correctamente.

- *Documentación.* Es importante, dado que los programas en assembly son más difíciles de comprender que los escritos en lenguajes de alto nivel. Es imprescindible, en concreto, documentar el uso de la memoria, de la pila (sobre todo al pasar parámetros) y de los registros dentro de las subrutinas.

- *Mantenimiento.* Si un programa va a ser utilizado después de un buen período de tiempo, de seguro necesitará alguna revisión, ya sea para enmendar errores, ya sea para hacer alguna que otra mejora. Aquí, en esta fase, es donde se valora el tiempo invertido en un diseño cuidadoso y en una buena documentación. Si el programa se diseñó y/o se documentó pobremente, será mejor que lo vuelva a escribir por completo y no intentar cambios.

Necesitamos ahora un proyecto para aplicar estas habilidades. Nada más apropiado, para nuestra primera aventura en assembly estructurado, que un monitor/depurador en código máquina. Si ya ha usado un ensamblador, le resultará familiar el tipo de utilidades que se esperan de un monitor/depurador. En esencia, proporcionará al programador el tipo de facilidades de edición que el programador en BASIC da por supuestas; es decir, la posibilidad de inspeccionar y cambiar los contenidos de la memoria.

En el próximo capítulo de este curso de lenguaje máquina trataremos este proyecto y recorreremos, desde la fase de diseño, todas las etapas descritas anteriormente, con la finalidad de crear una ayuda de programación importante y que le sea de auténtica utilidad.

Columna vertebral
No existen estructuras de control escritas en assembly, por eso puede ser útil imitar los métodos comprobados en lenguajes de alto nivel. Las estructuras que aquí mostramos son claras y elegantes en ambos lenguajes, de bajo y alto nivel, y se emplearán para excluir cualquier otra alternativa

Estructuras de control

Estructura IF...THEN...ELSE

Seudocódigo	Lenguaje assembly
	TRES FCB 3
	.....
IF NUM1 = 3	IF LDA NUM1
THEN	CMPA TRES
rutina1	BNE ELSE
	THEN .....
ELSE	*rutina1
rutina2	BRA FINIF
ENDIF	ELSE .....
	*rutina2
	FINIF .....

Estructura WHILE...WEND

Seudocódigo	Lenguaje assembly
WHILE NUM1 <=3	WHILE LDA NUM1
rutina a repetir	CMPA TRES
WEND	BGT FINWHILE
	*rutina a repetir
	BRA WHILE
	FINWHILE .....

Estructura REPEAT...UNTIL

Seudocódigo	Lenguaje assembly
REPEAT	REPEAT.....
rutina a repetir	*rutina a repetir
UNTIL NUM1 <3	LDA NUM1
	CMPA TRES
	BGE REPEAT
	UNTIL .....

Estructura FOR...NEXT

Seudocódigo	Lenguaje assembly
FOR NUM1=1 TO NUM2	LDA NUM2
rutina a repetir	FOR .....
NEXT NUM1	*rutina a repetir
	DECA
	BGT FOR
	NEXT .....

El efecto del defecto

"Deus ex machina" es un original juego que le permite al usuario asumir el papel de protagonista de una "fantasía televisiva totalmente animada"

Dado que los juegos por ordenador se han convertido en una parte importantísima de la industria del ocio, quizá era inevitable que las firmas de software aunaran sus esfuerzos con otros sectores del negocio del entretenimiento. Automata Software, empresa conocida por su serie de juegos de Piman, ha dado los primeros pasos en este camino desarrollando un producto que no sólo contiene software para ordenadores sino también una cassette de audio que se puede sincronizar con el programa para ordenador para proporcionarle al juego una banda sonora.

La idea sobre la que se sustenta *Deus ex machina*, cuyo desarrollo consumió seis meses y su programación otros tres, es que un ordenador omnipotente del futuro se rebela y ayuda a crear un "defecto" humano, un ser imperfecto. El jugador, como ser imperfecto, pasa a través de diversas etapas a lo largo del juego, que describen experiencias que abarcan desde la infancia hasta la vejez. La participación directa del jugador comienza en la concepción, guiando al espermatozoide hacia el óvulo. A medida que el niño va creciendo, está sujeto al ataque constante de la "policía del defecto". Está en manos del jugador desviar estos ataques, utilizando ya sea el teclado o la palanca de mando. Se consiguen puntos manteniendo el "porcentaje de entidad ideal", que empieza con un 99 % y va disminuyendo en función de los asaltos de la policía del defecto. Cuando el ser imperfecto alcanza la edad adulta, la naturaleza de esos ataques cambia y el jugador se debe adaptar a ellos.

Después de cargado el juego, se debe sincronizar la banda sonora con el programa. Al hacerlo se debe tener mucho cuidado, porque las distintas pantallas están cronometradas como para coincidir exactamente con las palabras y la música, y esto contribuye enormemente al entretenimiento que proporciona el paquete.

El programa se divide en dos segmentos, una mitad en cada lado de la cassette, que representan un total de 96 K de código. Al final del lado 1, después de una escena amorosa en la que el jugador debe desplazar el cursor por su cuerpo para recibir los besos que van flotando hacia él, se debe cargar el lado 2. No se debe apagar el ordenador y, nuevamente, debe ponerse atención en sincronizar la banda de sonido correctamente. La implicación del jugador en la segunda mitad consiste fundamentalmente en ir saltando obstáculos antes de llegar a la "vejez". En este punto aparecen en la pantalla grandes coágulos de sangre, que el jugador debe disolver. Al concluir el juego, independientemente del marcador, el ser imperfecto muere.

Deus ex machina es inusual por el hecho de que no hay ningún marcador ganador y, en realidad, el jugador ni siquiera necesita participar para nada en el juego. Los acontecimientos se suceden de la misma forma sin ninguna participación, de modo que el jugador tiene la opción de involucrarse en el desarrollo del juego o bien mirarlo como una fuente de entretenimiento. Los gráficos son excelentes e imaginativos. A pesar de que no hay ninguna pantalla que corte el aliento, sí reflejan todas el cuidado y la atención que se le dedicaron a todos los detalles del paquete en su conjunto.

La música de la banda sonora la escribió y la dirigió totalmente el cofundador de Automata, Mel Croucher, quien también escribió el guión. Las canciones son agradables, sin llegar a ser excepcionales. El mejor número es el que acompaña a la escena en el cual el ser imperfecto cobra vida, y lo interpreta Ian Dury.

El guión y la banda sonora son muy diferentes a los de la mayoría de los juegos por ordenador y reflejan la filosofía de no violencia a la que responden todos los juegos de Automata. Los usuarios que disfruten destruyendo hordas frenéticas de extraterrestres atacantes probablemente quedarán decepcionados, y a muchas personas el contenido semimístico de las letras de las canciones les resultará molesto. Sin embargo, se debe aplaudir de todo corazón a Automata por su innovadora idea. El programa es un valiente experimento y sin ninguna duda será considerado como un paso importante en el desarrollo del ocio informatizado.

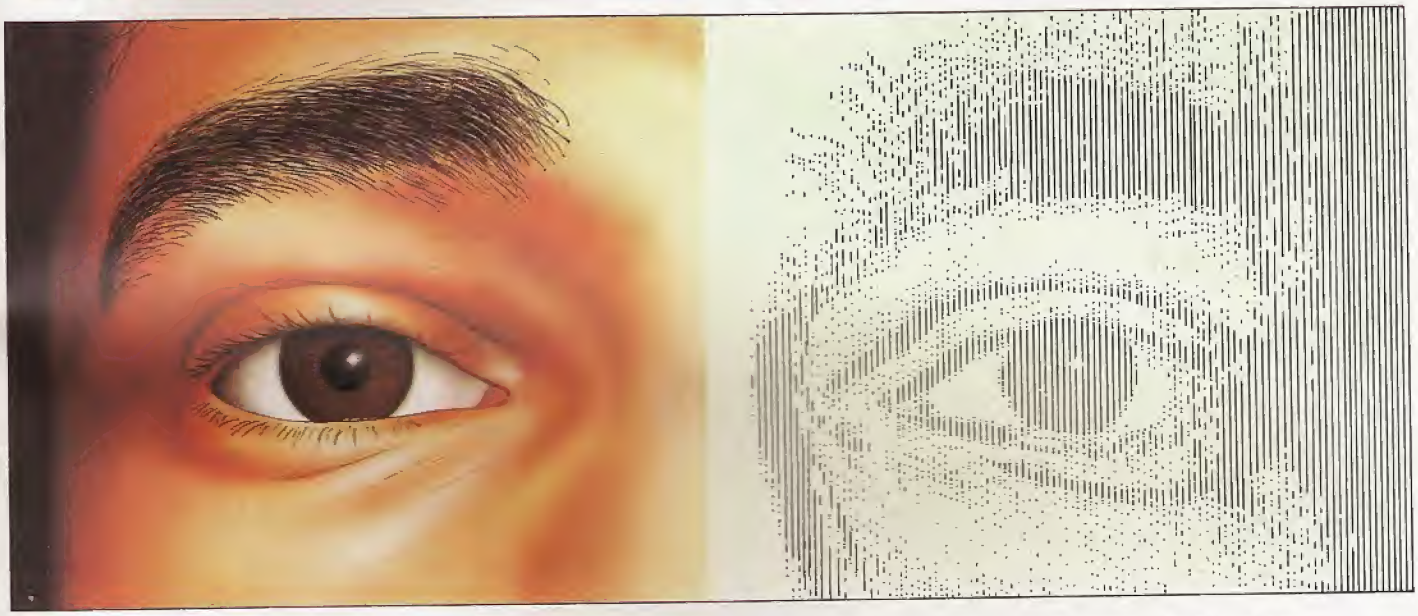
Desarrollo del juego
Deus ex machina es para jugar con él o para contemplarlo como fuente de entretenimiento. Posee una amplia variedad de visualizaciones, si bien algunas guardan cierto parecido entre sí. Las tácticas requeridas para mantener la "entidad ideal" cambian constantemente. El marcador se indica como porcentaje en el ángulo inferior derecho y va disminuyendo lentamente a medida que el juego (y la vida del ser imperfecto) avanza.

Ian McKinnell

Deus ex machina: Para el Spectrum de 48 K
Editado por: Automata Ltd, 27 Highland Road, Portsmouth, Hants, PO4 9DA, Gran Bretaña
Autores: Mel Croucher, Andrew Stagg
Palanca de mando: Opcional
Formato: Cassette

Los ojos del robot

Examinemos los problemas que implica preparar un robot para «ver» los objetos del mundo exterior



Steve Cross

La vista del robot y la vista del hombre
La naturaleza de la capacidad humana de ver se basa en gran medida en la interacción entre un complejo sistema de nervios y receptores, todos ellos procesados por el cerebro. Si bien una imagen visual se compone de patrones de luz y oscuridad marcados en la retina, el verdadero acto de "ver" tiene lugar en el cerebro. El cerebro de un robot procesa, asimismo, una imagen de patrones de luz y oscuridad, pero posee un grado de precisión muy inferior

De todos los sentidos, tal vez el más importante sea el de la vista. Las percepciones visuales son tan importantes para nuestra comprensión del mundo, que con frecuencia se utiliza la frase "es como tratar de describirle el color a un ciego" para ilustrar las dificultades que entraña explicar a alguien algo sobre lo que no tiene absolutamente ningún conocimiento. Sin la vista, nuestro conocimiento del mundo se ve seriamente restringido y, del mismo modo, un robot que no posea aparato visual está igualmente en situación de desventaja. Ya hemos visto cómo los robots utilizan sensores para detectar la presencia de un objeto en su camino; ahora deseamos desarrollar un sistema que los dote de un equipo visual tan eficaz como el del hombre.

El ser humano posee un iris que actúa a modo de lente, controlando la cantidad de luz que penetra en el interior del ojo, y una retina en la cual la lente focaliza la imagen. Pero el hecho es que el ojo en realidad no "ve" nada en absoluto; es sólo un transductor que convierte una señal en otra forma más aceptable. La verdadera tarea de ver la lleva a cabo el cerebro sobre la base de las señales que recibe desde sus sensores.

De modo que el tema de la vista del robot lo podemos dividir en dos partes claramente diferenciadas. La primera implica la construcción de un "ojo" adecuado que actúe como sensor para el sistema visual del robot; la segunda parte es el procesamiento de ordenador que se debe realizar antes de que el robot pueda darles algún significado a las señales provenientes de su sensor.

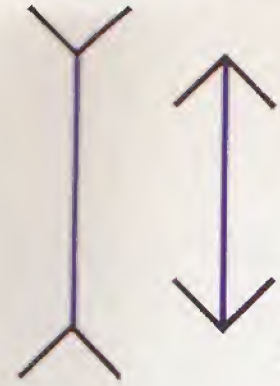
Construir un ojo robot no es demasiado difícil. En su nivel más simple, una célula fotoeléctrica puede actuar como una forma sencilla de ojo. Ésta puede dar una señal que corresponda a la iluminación global del campo de visión; como ya hemos visto, esto puede ser útil si deseamos simplemente que nuestro robot "se dirija" hacia una luz brillante o siga una línea blanca pintada sobre un fondo oscuro. El programa para emplear esta entrada sensorial puede, asimismo, ser simple, dado que la información recibida es limitada y que la cantidad de acciones que el robot puede emprender como consecuencia de dichas señales simples es proporcionalmente restringida.

Pero a esto apenas si podemos denominarlo "vista", en la acepción literal del término. Específicamente, necesitamos un sistema visual que pueda construir una imagen bidimensional completa del mundo, permitiendo que el "cerebro" del robot examine exactamente la misma información que procesa el cerebro humano.

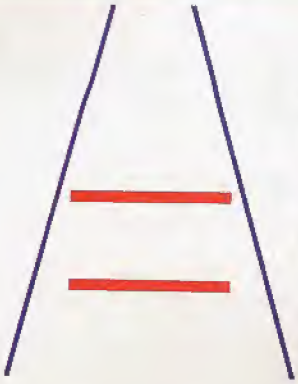
Una respuesta a este problema utiliza una única célula fotoeléctrica con una lente colocada frente a ella. Ésta explora la zona de la imagen por delante del robot, barriendo mecánicamente todo el campo de visión hasta construir una imagen completa; ésta, entonces, se puede almacenar en la memoria del ordenador. En la práctica, lamentablemente, este método es lento y poco fiable.

En la mayoría de los casos, sin embargo, el ojo robot se compone de algún tipo de cámara de video. Esta cámara puede ser del tipo estándar que se utiliza para las transmisiones de televisión, o puede ser un dispositivo especializado diseñado específicamente para la visión robótica. Algunos aparatos de esta última clase emplean chips especiales denominados *RAM ópticas*; éstos se componen de memoria RAM en la que el valor de cada byte se establece automáticamente en función de la cantidad de luz que cae sobre ese byte determinado.

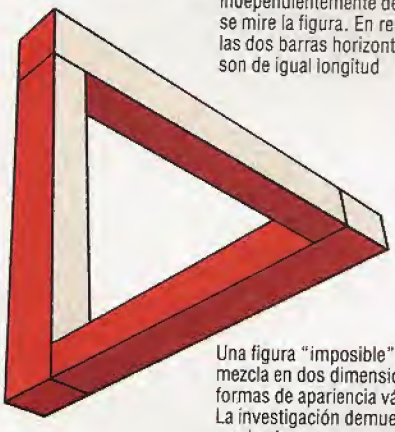
Lo que uno cree que ve



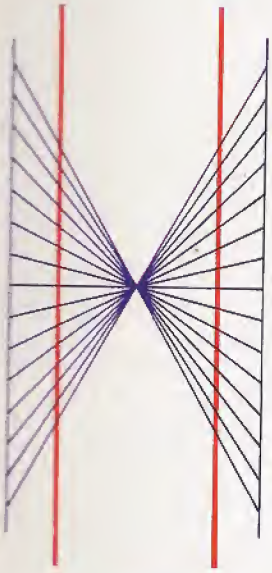
¿Cuál es más larga? La ilusión de Muller-Lyer hace que uno crea que la línea vertical de la figura de la izquierda es más larga que la línea vertical de la de la derecha. En realidad, las dos tienen la misma longitud



En la ilusión de la vía, la barra horizontal superior parece ser más larga, independientemente de cómo se mire la figura. En realidad, las dos barras horizontales son de igual longitud

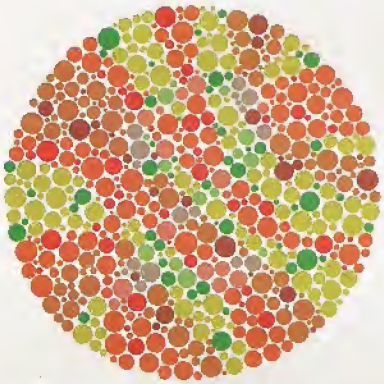


Una figura "imposible", que mezcla en dos dimensiones formas de apariencia válida. La investigación demuestra que los humanos son incapaces de reconocer esto como un objeto



A pesar de que las líneas verticales parecen estar combadas, en realidad son rectas. El ojo inclina la imagen para adaptarla al despliegue de los rayos

Kevin Jones



Los patrones de color ilustrados, tomados de los tests de Ishihara para la ceguera de colores, miden deficiencias de rojo-verde. En el ejemplo de la izquierda, las personas con visión normal verán el número 74, mientras que quienes adolezcan de una deficiencia de rojo-verde, verán el número 21. En el círculo de la derecha, la visión normal no discernirá nada, o sólo un 2 difuso, mientras que quienes padezcan una deficiencia de rojo-verde distinguirán claramente un 2

Estos dispositivos se están abaratando cada vez más y proporcionan una zona de memoria RAM que contiene toda la información relativa a la escena captada por el ojo del robot.

En general, la salida de un ojo robot se retiene en una matriz bidimensional, cada elemento de la cual contiene un valor que corresponde al brillo de la luz que cae sobre esa parte concreta de la escena que se está contemplando. La cantidad de elementos de la matriz proporciona la *resolución* de la imagen, y la gama de números que se puede retener en cada elemento de la matriz determina la cantidad de niveles de la *escala de grises* que se pueden discernir. Tradicionalmente, en los sistemas de visión cada elemento de la matriz se denomina *pixel* o *elemento de imagen*. De modo que una matriz de imagen de 500 por 250 pixels que represente niveles de brillo asignándole un byte a cada pixel tendría una resolución horizontal de 500 pixels, una resolución vertical de 250 pixels, un total de 125 000 pixels y 256 escalas de grises desde el negro al blanco puro. Para dar una idea del detalle que proporcionaría tal imagen, considere una imagen de televisión estándar. El sistema español utiliza 625 líneas verticales, de modo que la resolución vertical es de 625 pixels. Para obtener una resolución similar en sentido horizontal, se necesitarían aproximadamente 1 000 pixels (porque la pantalla es más ancha que alta), y los niveles de escala de grises se podrían representar mediante el mismo único byte para proporcionar 256 niveles de brillo. Un sistema robot con una resolución equivalente daría una imagen aceptable para que la procesara el ordenador.

Los procesos que debe realizar el "cerebro" robot con el objeto de "ver" esta imagen siguen unas pautas establecidas. El primer paso supone ajustar los niveles de escala de grises de modo que los pixels adyacentes con niveles similares de escala de grises se "emparejen" al mismo nivel. El ordenador trabaja sobre la superficie completa de la imagen uniformizando los niveles para eliminar cualquier pequeña irregularidad. Una vez hecho esto, el ordenador vuelve a examinar la imagen, percibiendo todos los pixels adyacentes que poseen niveles de escala de grises notoriamente diferentes; estas diferencias se enfatizan entonces. La finalidad de esto último es lograr que las características importantes de la imagen se marquen con límites, tales como líneas y bordes, que aparezcan como cambios súbitos en el nivel de escala de grises: el ordenador toma nota de los mismos, enfatizándolos para asegurar que sobresalgan.

Después de realizado esto, el ordenador explora de nuevo la imagen en busca de todos los cambios verdaderamente significativos en los niveles de escala de grises. Luego utiliza los mismos de forma muy parecida a como resolvería un ser humano uno de esos acertijos gráficos que aparecen en periódicos y tebeos que consisten en unir correlativamente una serie de puntos para componer una imagen. En la mayoría de estos juegos se cuenta con la ayuda que supone el hecho de que los puntos estén numerados; el ordenador no dispone de tal ayuda y simplemente debe seguir lo que parezca una ruta. Al final de este proceso el robot tendrá una imagen interna de la escena en la cual habrá "suavizado" la imagen y dibujado líneas alrededor de aquellos objetos que parezcan ser importantes.

Pero ¿es esto "ver"? En realidad, todo lo que ha hecho el robot es llevar a cabo ciertas transformaciones de la escena; en otras palabras, todavía no "sabe" qué es lo que está mirando.

Existen dos soluciones para este problema. La primera consiste en programar al robot con un conjunto de reglas que se expresan como una serie de sentencias simples acerca del mundo visual. Esto se conoce como aproximación *de abajo arriba* (*bottom-up*) a la percepción visual, frase que alude al hecho de que el robot empieza con cosas muy simples y a partir de ellas intenta evaluar lo que está viendo en un nivel de comprensión más complejo. El segundo enfoque es el de programar al

Reconocer objetos

Para aprender un objeto, un robot sigue un proceso similar al aprendizaje humano: toma la imagen que ve y la compara con la imagen de un objeto conocido hasta hallar una pareja. A diferencia de los humanos, sin embargo, los robots poseen una capacidad visual limitada y sólo pueden almacenar una gama de patrones de objetos muy restringida. Al no contar con la profundidad de la experiencia humana y las capacidades comparativas tan desarrolladas del hombre, el robot no puede determinar que el objeto percibido corresponde a los perfiles almacenados conocidos como "teléfono", ni siquiera haciendo rotar su imagen en varias posiciones

PLANO: SILUETA

OBJETO

ELEVACIÓN FRONTAL: SILUETA

ELEVACIÓN LATERAL: SILUETA

Kevin Jones

robot con un conjunto de objetos que es probable que vea y dejar luego que examine la imagen para ver si alguno de estos objetos está presente. Esto se denomina aproximación *de arriba abajo* (*top-down*) en razón de que el robot empieza con una idea de alto nivel muy compleja acerca de lo que podría estar viendo y luego comprueba si se corresponde con su verdadera entrada visual.

Para ilustrar la diferencia entre los dos métodos, considere un robot que esté mirando una mesa. La aproximación de abajo arriba consiste en analizar la imagen y descubrir que contiene cuatro partes verticales y, junto a la parte superior de las mismas, una gran superficie horizontal. Esto corresponde al conocimiento preprogramado de que podría haber una gran superficie descansando sobre cuatro patas y que esta estructura sería lo que se denomina una mesa. La aproximación de arriba abajo empezaría con el robot mirando a la mesa e interrogándose: "¿Es esto una mesa?" Esta pregunta se la puede formular porque posee un modelo interno de una mesa con el cual comparar su entrada visual.

En general, la aproximación de abajo arriba capacita al robot para ver cosas con las que nunca se ha encontrado y para entender algo acerca de ellas; pero para hacer esto precisa una gran cantidad de programación detallada que le proporcione las reglas básicas necesarias acerca del mundo con el cual se encontrará. Sin embargo, la aproximación de arriba abajo permite que el robot sólo reconozca objetos sobre los que ya posea algún conocimiento interno; de modo que cualquier cosa nueva supondría problemas.

Los diseñadores de robots utilizan ambos métodos y en ocasiones los combinan. Parece probable que los humanos empleemos procedimientos similares para la percepción visual; no obstante, lo hacemos automáticamente y no somos conscientes de que estos procesos se están realizando.

Pero la visión del robot, de momento, dista mucho de ser perfecta. Ello se debe a varios motivos. Uno de los más importantes es la inmensa cantidad de potencia de proceso que se necesita para procesar una imagen. Recuerde que el sistema de nuestro ejemplo tenía 125 000 pixels almacenados cada uno como un byte: por consiguiente, se deberían procesar más de 122 K de memoria para cada imagen. Si bien hemos simplificado nuestra descripción, muchos de los procesos que se deben llevar a cabo en cada pixel son bastante complejos desde el punto de vista matemático. Si el robot ha de observar el mundo que lo rodea en "tiempo real" (es decir, considerar los acontecimientos a medida que se van produciendo), entonces se reciben 25 imágenes diferentes por segundo (esto también es así en el caso de las cámaras de televisión). Ello significa que el robot necesitaría procesar más de 3 050 K de datos por cada segundo, ¡lo que equivale aproximadamente al contenido de más de una docena de discos flexibles!

Para tratar el problema del proceso se pueden considerar dos enfoques. Uno consiste en desarrollar hardware para fines especiales que efectúe el procesamiento de las imágenes (este tipo de hardware está comenzando a aparecer en la actualidad). Por otra parte, la resolución de la imagen y la cantidad de niveles de escala de grises se podrían reducir para adaptarse al hardware existente. Esto conduciría a que la imagen se procesara más rápidamente, pero la calidad de ésta sería inferior.

En estos momentos, sin embargo, el tema de la visión del robot todavía no se entiende del todo, al menos no más de lo que se comprenden los detalles acerca de la visión del hombre. Cuando utilizan un sistema de visión, los robots a menudo cometen errores. Bien podría suceder que, finalmente, la única respuesta fuera desarrollar sistemas en los cuales el robot "aprendiera" a ver cosas en vez de programarlo detalladamente sobre qué es lo que puede y no puede ver. Y a la larga podría darse el caso de que nunca pudiera "ver" las cosas correctamente hasta el desarrollo de una forma que le proporcionase un conocimiento muchísimo mayor acerca del mundo circundante.

Todas cambian

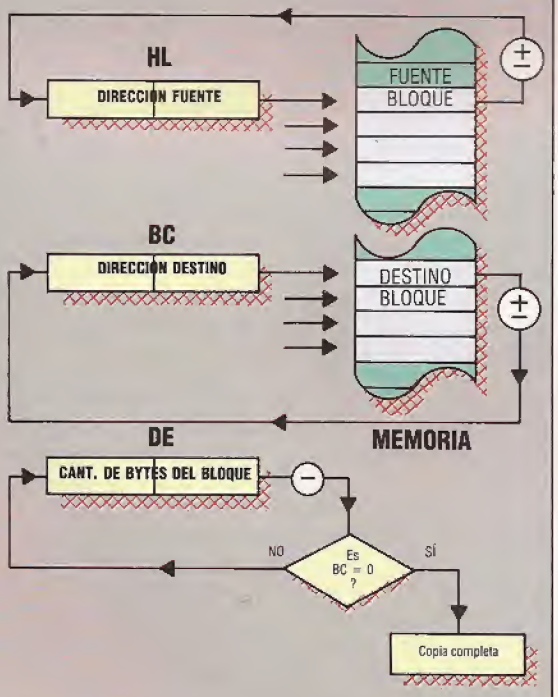
En el Spectrum es distinta la implementación del programa de sustitución de variables: la utilidad se mezcla al final del programa

A medida que el programa de sustitución de variables explora a través del programa, va haciendo una copia de la versión modificada en una zona por encima de RAMTOP. La versión modificada se vuelve a copiar entonces en la zona del programa principal mediante un programa en código máquina que regula la cantidad de espacio disponible si se ha alterado la longitud del programa, de modo que la nueva versión quepa en la zona para BASIC.

La primera parte del programa en BASIC es similar al programa de búsqueda de variables (véase p. 1145). Hay algunas variables extras, incluyendo Altprog, que señala el comienzo de la zona reservada para la copia del programa, y Altapunt, que lleva el registro de a dónde debe ir el siguiente byte del programa modificado. Los cambios principales implican copiar el programa, en vez de simplemente leerlo. El copiado se realiza mediante la subrutina

Escritura automática

Los opcodes LDIR y LDDR del Z80 son instrucciones para transferencia de bloques que utilizan el incremento o el decremento automáticos: el registro HL se inicializa para apuntar al comienzo del bloque fuente o emisor, DE debe apuntar al comienzo del destino o receptor, y BT debe contener la cantidad de bytes del bloque. LDIR y LDDR copian entonces el byte fuente en el byte de destino, incrementando o decrementando automáticamente HL y DE, y decrementando BC hasta que llega a cero, cuando se considera completa la copia. Observe que LDIR es una copia «tonta» (véase p. 1206): se da por supuesta la inteligencia del programador



Kevin Jones

Programa de sustitución de variables

```
9000 INPUT "Nombre a buscar?"; LINE t$
9005 INPUT "Reemplazar por?"; LINE r$
9010 FOR i=1 TO LEN (t$)
9020 IF t$(i)>="a" AND t$(i)<="z" THEN LET t$(i)=CHR$ (CODE
     (t$(i)) - 32)
9030 NEXT i
9040 LET Distint REM=234
9050 LET Comillas=34
9060 LET Nuevalinea=13
9070 LET Subrayado=95
9080 LET Numero=14
9090 LET PROG=23635
9100 LET Apuntexto=PEEK (PROG)+256*PEEK (PROG+1)
9102 LET Altprog=46000
9105 LET Altapunt=Altprog
9110 LET Numlinea=256*PEEK (Apuntexto)+PEEK (Apuntexto+1)
9111 PRINT Numlinea
9120 IF Numlinea >=9000 THEN GO TO 9600
9130 LET q=2:GO SUB 9800
9135 LET Longdir=Altapunt
9140 LET Siglinea=Apuntexto+2+PEEK (Apuntexto)+256*PEEK
     (Apuntexto+1)
9150 LET q=2:GO SUB 9800
9160 LET Byte=PEEK (Apuntexto):LET q=1:GO SUB 9800
9170 IF Byte=Nuevalinea THEN GO TO 9110
9180 IF Byte<>Distint REM THEN GO TO 9220
9190 REM Copiar REM sin alterar
9200 LET q=Siglinea-Apuntexto:GO SUB 9800
9210 GO TO 9110
9220 IF Byte<>Comillas THEN GO TO 9280
9230 REM Copiar todo lo que este entre comillas, pero parar al final
     de la linea en caso de comillas sin cerrar
9235 LET q=1
9240 IF PEEK (Apuntexto+q-1)=Nuevalinea THEN GO SUB
     9800:GO TO 9110
9250 IF PEEK (Apuntexto+q-1)=Comillas THEN GO SUB 9800:GO
     TO 9160
9260 LET q=q+1
9270 GO TO 9240
9280 REM Copiar numero binario de 5 bytes
9290 IF Byte=Numero THEN LET q=5:GO SUB 9800:GO TO 9160
9310 REM El primer caracter del nombre debe ser una letra en
     mayuscula o minuscula
9320 IF Byte>=CODE ("A") AND Byte<=CODE ("Z") THEN LET
     c$=CHR$ (Byte):GO TO 9370
9330 REM Usar mayusculas en vez de minusculas
9340 IF Byte>=CODE ("a") AND Byte<=CODE ("z") THEN LET
     c$=CHR$ (Byte-32):GO TO 9370
9360 GO TO 9160
9370 LET n$=""
9380 LET n$=n$+c$
9400 REM Letra, digito o subrayado despues del primer caracter del
     nombre
9410 IF PEEK (Apuntexto)>=CODE ("A") AND PEEK
     (Apuntexto)<=CODE ("Z") THEN LET c$=CHR$ (PEEK
     (Apuntexto)):LET Apuntexto=Apuntexto+1:GO TO 9380
9420 REM Usar mayusculas en vez de minusculas
9430 IF PEEK (Apuntexto)>=CODE ("a") AND PEEK
     (Apuntexto)<=CODE ("z") THEN LET c$=CHR$ (PEEK
     (Apuntexto)-32):LET Apuntexto=Apuntexto+1:GO TO 9380
9440 IF PEEK (Apuntexto)>=CODE ("Ø") AND PEEK
     (Apuntexto)<=CODE ("9") THEN LET c$=CHR$ (PEEK
     (Apuntexto)):LET Apuntexto=Apuntexto+1:GO TO 9380
9450 IF PEEK (Apuntexto)=Subrayado THEN LET c$=CHR$ (PEEK
     (Apuntexto)):LET Apuntexto=Apuntexto+1:GO TO 9380
9460 REM Terminar con $ para variable en serie
9470 IF PEEK (Apuntexto)=CODE ("$") THEN LET n$=n$+"$":LET
     Apuntexto=Apuntexto+1:GO TO 9500
9480 REM (si matriz o funcion
9490 IF PEEK (Apuntexto)=CODE ("(") THEN LET n$=n$+CHR$
     (PEEK (Apuntexto)):LET Apuntexto=Apuntexto+1
9500 IF n$=t$ THEN LET n$=r$
9505 LET Altapunt=Altapunt-1
9510 FOR p=1 TO LEN (n$)
9520 POKE Altapunt,CODE (n$(p))
9530 LET Altapunt=Altapunt+1
9540 NEXT p
9550 IF n$<>r$ THEN GO TO 9160
9560 LET Longbaja=PEEK (Longdireccion)+LEN (r$)-LEN (t$)
9570 IF Longbaja>255 THEN LET Longbaja=Longbaja-256: POKE
     Longdireccion+1,1+PEEK (Longdireccion+1)
9580 POKE Longdireccion, Longbaja
9590 GO TO 9160
```

```
9599 REM Prepararse para desplazar programa alterado nuevamente
     a zona programa principal
9600 LET Antigualong=Apuntexto-(PEEK (PROG)+256*PEEK
     (PROG+1))
9610 LET Nuevalong=Altapunt-Altprog
9620 POKE 45060,Nuevalong-256*INT (Nuevalong/256)
9630 POKE 45061,INT (Nuevalong/256)
9660 POKE 45055,Apuntexto-256*INT (Apuntexto/256)
9670 POKE 45057, INT (Apuntexto/256)
9680 IF Antigualong=Nuevalong THEN RANDOMIZE USR (45084)
9690 IF Antigualong<Nuevalong THEN LET X=Nuevalong-
     Antigualong
9700 IF Antigualong>Nuevalong THEN LET
     X=Apuntexto-(Antigualong-Nuevalong)
9710 POKE 45058,X-256*INT (X/256)
9720 POKE 45059,INT (X/256)
9730 IF Antigualong<Nuevalong THEN RANDOMIZE USR 45062
9740 IF Antigualong>Nuevalong THEN RANDOMIZE USR 45074
9800 FOR p=Apuntexto TO Apuntexto+q-1
9810 POKE Altapunt,PEEK (p)
9820 LET Altapunt=Altapunt+1
9830 NEXT p
9840 LET Apuntexto=p
9850 RETURN
9900 SAVE "SUSTITUCION" LINE 9910:SAVE "SUST MC"CODE
     45064,37: STOP
9910 CLEAR 45055: LOAD "SUST MC"CODE
```

Cargador de código máquina

```
 10 CLEAR 45055
 20 LET a=45062
 30 FOR I=1000 TO 1040 STEP 10
 40 LET s=0
 50 FOR a=a TO a+7
 60 READ b
 70 POKE a,b
 80 LET s=s+b
 90 NEXT a
100 READ c
110 IF s<>c THEN PRINT"ERROR DE DATO EN LINEA";I: STOP
120 NEXT I
1000 DATA 42,0,176,237,75,2,176,205,913
1010 DATA 85,22,24,10,42,0,176,237,596
1020 DATA 91,2,176,205,229,25,33,176,937
1030 DATA 179,237,91,83,92,237,75,4,998
1040 DATA 176,237,176,207,255,0,0,0,1051
```

Programa de sustitución en assembly

```
0000                HCSITI  EQU     $1655
0000                RCLAM1  EQU     $19E5
0000                TO      EQU     $B000
0000                T1      EQU     $B002
0000                T2      EQU     $B004
0000                ALTPRG  EQU     $B3B0
0000                PROG    EQU     $5C53
B006                        ORG     $B006
B006 2A00B0         UP      LD      HL, (T0)
B009 ED4B02B0               LD      BC, (T1)
B00D CD5516                 CALL    HCSITI
B010 180A                   JR      COPY
B012 2A00B0         DOWN    LD      HL, (T0)
B015 ED5B02B0               LD      DE, (T1)
B019 CDE519                 CALL    RCLAM1
B01C 21B0B3         COPY    LD      HL, ALTPRG
B01F ED5B535C               LD      DE, (PROG)
B023 ED4B04B0               LD      BC, (T2)
B027 EDB0                   LDIR
B029 CF                     RST     8
B02A FF                     DB      $FF
```

de la línea 9800, que copia la cantidad de bytes especificada por q y actualiza los apuntadores de los programas antiguo y nuevo.

Los nombres de las variables se copian mediante el código que comienza en la línea 9500 y, si el nombre de la variable se ha modificado, se alteran los dos bytes del principio de la línea que retienen la longitud de la línea, para reflejar el cambio en la longitud.

Después de que todo el programa se ha modificado y se ha copiado en la nueva zona, el programa BASIC calcula los valores que necesita el código máquina para volver a copiar el programa alterado, y luego coloca (POKE) estos valores en las posiciones de memoria donde el código máquina espera hallarlas. Si el programa nuevo y el antiguo tienen la misma longitud, el nuevo programa se puede copiar en el mismo espacio que ocupa el programa antiguo. En este caso, la única información que precisa el programa en código máquina es la longitud del programa.

Si el programa nuevo es más largo que el antiguo, necesitamos hacer espacio extra en la zona de programas desplazando hacia arriba la rutina de sustitución de variables, que deseamos conservar. El espacio extra se consigue llamando a la subrutina de ROM, HACER-SITIO, en la dirección 1655 hexa. Cuando se llama a HACER-SITIO, el registro doble HL debe contener la dirección de después del lugar donde se ha de hacer el espacio, y el registro doble BC debe contener la longitud del espacio necesario. El valor requerido para HL no es más que el valor final de Apuntexto, y el valor de BC es la diferencia entre la longitud antigua y la nueva.

Si el programa nuevo es más corto que el antiguo, hemos de desplazar hacia abajo el programa de sustitución de variables. Esto lo podemos hacer utilizando la subrutina de ROM RECLAMAR-1 de la dirección 19E5 hexa. Cuando se llama a RECLAMAR-1, el registro doble HL debe contener la dirección del primer byte a dejar solo, y el registro doble DE debe contener la dirección del primer byte a reclamar. El valor requerido para HL es, nuevamente, el valor final de la variable Apuntexto, y el valor requerido para DE se calcula restándole a Apuntexto la diferencia entre la longitud antigua y la nueva.

El programa alterado se vuelve a copiar en la zona del programa principal mediante la instrucción para movimiento de bloques LDIR (*LoaD with Increment and Repeat*). La dirección de comienzo de la zona del programa alterado se carga en HL, la dirección de comienzo de la zona del programa principal en DE, la longitud del programa alterado en BC, y después la instrucción LDIR desplaza todo el programa alterado, byte a byte.

Las dos últimas líneas del programa en lenguaje assembly utilizan otra rutina de ROM, en la dirección 8. Esta es una rutina de «informe» que imprime un mensaje de error y otros comentarios. La rutina es llamada por la instrucción RST 8, y el informe producido se especifica por el byte que sigue a la instrucción RST 8. El valor del byte es uno menos que el número del informe, de modo que FF hexa, o −1, da el OK o el informe de Programa terminado; 0 da NEXT sin FOR, y así sucesivamente. El programa en código máquina termina con RST8, en vez de la habitual instrucción RET, para evitar el retorno al programa en BASIC que se ha desplazado.

Dando el tono

Veamos otros dos parámetros de la sintetización de sonido: volumen y altura

El volumen de un tono está determinado por la escala de oscilación de la forma de onda que genera el tono. Dicho en otras palabras, el volumen depende de la diferencia entre el valor máximo de la forma de onda y el valor mínimo. Esta propiedad de una onda sonora se denomina *amplitud*.

Utilizando un sencillo programa en BASIC para hacer oscilar valores colocados en el registro de la puerta para el usuario podemos demostrar cuán fácilmente se puede controlar la amplitud de una forma de onda digital.

```
 10 REM **** AMPLITUD/FRECUENCIA BAS CBM ****
 20 :
 25 RDD=56579:REGDAT=56577
 30 POKE RDD,255:REM TODAS SALIDA
 40 FOR I=255 TO 0 STEP-15
 50 FOR J=1 TO 100
 60 POKE REGDAT,I:POKE REGDAT,0
 70 NEXT J,I

800 REM **** PROGRAMA MUESTRA ****
805 :
806 UPS=CHR$(145)
810 DIV=49798:REM POSICION FACTOR AMPLITUD
820 DEL=49799:REM POSICION FACTOR DEMORA
830 TME=49800:REM FACTOR DURACION
840 CALL=49801:REM DIRECCION COMIENZO PROGRAMA
850 :
860 RDD=56577:POKERDD,255:REM TODAS SALIDA
870 :
880 PRINTCHR$(147):REM LIMPIAR PANTALLA
890 PRINT:INPUT"FACTOR DE AMPLITUD 0-7";FA
900 IF FA<0 OR FA>7 THEN PRINT UP$;UP$;:GOTO890
910 POKE DIV,FA
920 :
930 PRINT:INPUT"FACTOR DE DEMORA 1-101";FD
940 IF FD<0 OR FT>101THEN PRINTUP$;UP$;:GOTO960
960 POKE TME,FT
990 :
1000 SYS CALL
1010 GETAS:IFAS="" THEN 1010
1020 IFAS="X" THEN 880:REM RECOMENZAR
1030 GOTO 1000:REM OTRO BEEP
```

Al principio del programa se genera una forma de onda tosca que oscila entre 255 y 0. Ello significa que la amplitud de la onda es 255. A medida que se va ejecutando el programa, el valor superior colocado en el registro de datos se reduce en pasos de 15. A medida que disminuye el valor superior va disminuyendo igualmente la amplitud; y el efecto de esto, al escuchar el sonido producido a través de un amplificador estéreo o un par de auriculares, es que el volumen del tono se va reduciendo gradualmente hasta desaparecer. De modo que el volumen de un tono sintetizado digitalmente se puede controlar limitando la escala de valores colocada en el registro de datos de la puerta para el usuario.

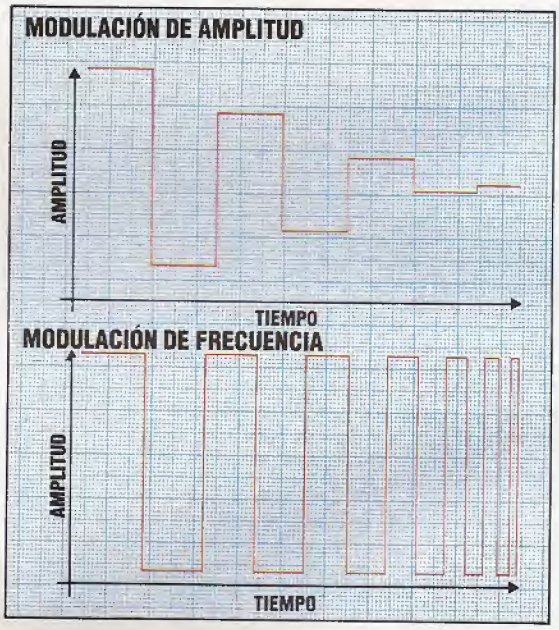
La altura de una nota está gobernada por la frecuencia de la onda generadora; o sea, el número de ciclos de formas de onda por segundo: cuanto mayor es el número de formas de onda producido por unidad de tiempo, más alta es la altura de la nota oída.

La frecuencia se puede controlar digitalmente de dos maneras principales: la primera es incrementando la frecuencia desde un límite inferior, tomando menos muestras de la forma de onda. Si una onda se dividiera en 100 muestras, por ejemplo, a un programa en código máquina le llevaría un cierto tiempo colocar cada valor en sucesión en el registro de datos, y, por lo tanto, se podría producir cierta cantidad de formas de onda completas por segundo. La cantidad de muestras, obviamente, determina la frecuencia del tono escuchado. Para duplicar la frecuencia, el programa en código máquina podría, en cambio, tomar de la tabla de datos que define la onda sólo un valor de cada dos. La frecuencia se podría triplicar tomando un valor de cada tres, y así sucesivamente. Este método tiene dos desventajas. La primera es que resulta difícil hacer pequeños ajustes de frecuencia sin distorsionar la forma de la onda. En segundo lugar, a medida que la frecuencia aumenta, la onda generada tiene cada vez menos relación con la forma de onda original, dado que se utilizan menos muestras.

Un método alternativo consiste en empezar con un bucle que vaya saltando a través de los datos de la forma de onda lo más rápidamente posible, proporcionando, por lo tanto, una frecuencia máxima. La frecuencia se puede, entonces, ajustar insertando en el bucle pequeñas demoras. Esto nos da un control mucho más preciso sobre la frecuencia del tono, pero significa que el número de muestras que componen la forma de onda debe ser pequeño si se ha de obtener una frecuencia máxima razonablemente elevada. Podemos emplear el segundo de estos dos métodos para producir un programa en código máquina que nos permita controlar tanto la frecuencia como el volumen.

El mejor método de reducir la amplitud de la forma de onda, conservando al mismo tiempo la

Construcción modular
Las formas de onda se pueden modular ya sea por frecuencia o por amplitud. La frecuencia altera la altura del tono oído y está determinada por el número de ciclos de forma de onda producidos por segundo. La amplitud altera el volumen del tono y es la diferencia entre los valores máximo y mínimo de un ciclo. Los diagramas ilustran cómo se puede modular la amplitud de modo que el volumen del tono producido vaya disminuyendo gradualmente y cómo se puede modular la frecuencia para producir un tono cuya altura se eleve



forma global de la onda, consiste en dividir cada valor de la tabla de la forma de onda en función de una constante. Esto se puede hacer de dos maneras: después de cargar en el acumulador cada valor pero antes de colocar el valor en el registro de datos, o antes de entrar en el bucle principal del programa. El primer método aumentará la cantidad de tiempo requerida para ejecutar cada ciclo del bucle principal y, puesto que este factor limita la frecuencia máxima a obtener, debemos optar por el segundo método. A partir de la tabla de forma de onda original se produce una segunda tabla, dividiendo por una constante cada valor tomado de la tabla original, colocando luego el resultado en la nueva tabla. Ésta es utilizada, entonces, por el bucle principal del programa para los datos de forma de onda. El método de división empleado es burdo. Una constante de amplitud determina la cantidad de veces que el valor de la tabla se desplaza hacia la derecha. Puesto que cada desplazamiento hacia la derecha es una división entera por dos, el efecto de utilizar un factor de amplitud de n es el de dividir cada tabla por $2n$.

Cuando el factor de amplitud es cero se utiliza la tabla original, y el programa se modifica a sí mismo para especificar la dirección base de ésta, en vez de la de la tabla de valores divididos.

La ejecución del bucle principal del programa se puede demorar mediante la inserción de un pequeño trozo de código cuya única misión es dejar transcurrir el tiempo mientras se lo ejecuta. Normalmente esto se hace decrementando o bien un número de ocho bits en uno de los registros índice, o un número de 16 bits de la memoria, desde un valor de demora dado hasta cero. Veremos que si calculamos las demoras máximas logradas mediante estos dos métodos, la disminución de un número de ocho bits proporcionará la demora suficiente.

El problema fundamental no es el de proporcionar suficiente demora (es decir, producir la frecuencia más baja), sino proporcionar la demora mínima; dicho en otras palabras, producir la frecuencia máxima. En la página 1212 utilizamos una forma de onda dividida en 80 pasos. El código extra requerido para la demora retrasa tanto el tiempo de ejecución, que ya no resulta práctico tener esta cantidad de pasos. Debajo se indica el código de que consta el bucle principal.

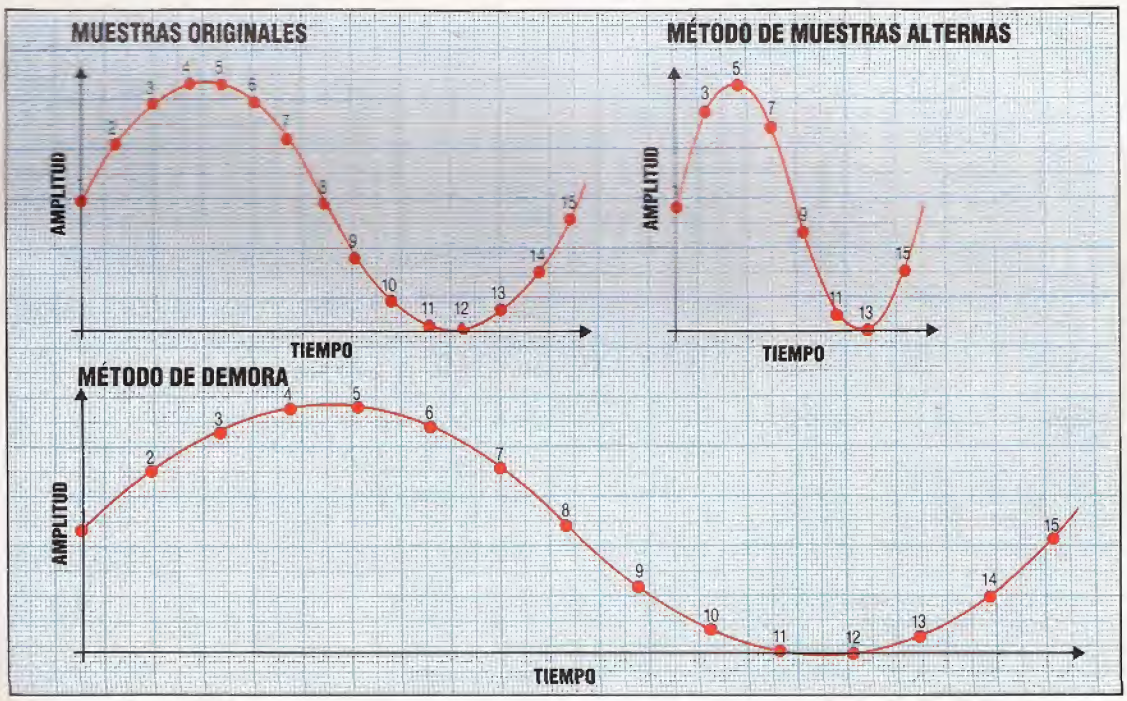
```
PRAL                                  Tiempo en ciclos de máq.
          LDX # $00                   2
SIGVAL
          LDA AMPTAB,X                4
          LDY DEMORA                  4
MASDEM
          DEY                         2
          BNE MASDEM                  3 (2 si fracasa el bucle)
          STA PUERTA                  4
          INX                         2
          CPX # PASOS                 2
          BEN SIGVAL                  3 (2 si fracasa el bucle)
```

El número total de ciclos de máquina requeridos viene dado por: $2+(4+4+(2+3)\times demora-1+4+2+2+3)\times pasos-1=1+(18+5\times demora)\times pasos$: y un valor mínimo de 1 produce la frecuencia máxima: $frec\ máx=1000000\div(1+23\times pasos)$.

Para una frecuencia máxima de alrededor de 3 000 Hz, esta fórmula indica que el número de pasos es 15. Ésta es la cantidad de muestras de la forma de onda que debemos utilizar para producir una frecuencia máxima razonable. Empleando 15 pasos, la fórmula original se puede escribir como: n.° de ciclos de máquina = $271+75\times demora$.

Si requerimos una frecuencia mínima de, supongamos, 128 Hz (alrededor de dos octavas por debajo de do central), entonces el valor de demora será 101. Este valor puede ser mantenido y decrementado en un registro índice.

El último problema producido por una frecuencia alterada es que, para un número dado de repeticiones del bucle de demora, la duración del tono producido disminuirá a medida que aumente la frecuencia. Ello se debe a que, puesto que ésta aumenta, ejecutar el bucle principal lleva menos tiempo. Para equilibrar esto debemos incluir un cálculo que establezca la cantidad de repeticiones requeri-



Liz Dixon

Estirando la serpiente
La frecuencia de una onda muestreada digitalmente se puede alterar tomando menos muestras o insertando una demora entre cada valor de muestra. Si la tabla original para la onda contiene 15 muestras de la onda, la frecuencia de la onda de salida se puede duplicar tomando sólo muestras alternas. Por otra parte, se pueden producir todos los valores, los 15, insertando una demora para duplicar el tiempo invertido en producir la onda completa, reduciendo a la mitad la frecuencia. El primer método permite utilizar muchas muestras en frecuencias inferiores pero sólo admite un control burdo de frecuencia. Con el segundo procedimiento es posible lograr un control de frecuencia mucho más fino, pero significa que se pueden utilizar menos muestras

das para producir un tono para una duración dada, independientemente de cuál sea la frecuencia de los tonos. Si hacemos que la unidad de duración sea de 1/50avo de un segundo, el número de repeticiones para un valor de demora dado sería: 1000000÷(50×(271+75×demora). Calcular esto en código máquina sería difícil y lento, de modo que prepararemos desde BASIC una tabla que ofrezca un valor tomado de la fórmula anterior para cada valor de la demora (o sea, de 1 a 109). La rutina en código máquina entonces ha de buscar el valor de repetición y almacenarlo en un decrementador, para producir un tono que dure 1/50avo de segundo.

Los usuarios de Commodore que posean un ensamblador pueden entrar el listado del código fuente y ensamblarlo, para producir un programa objeto que se pueda guardar. El programa de llamada volverá a cargar el programa objeto desde disco o cassette y preparará las tablas. Entrelo y ejecútelo. Digite NEW y entre el programa muestra en BASIC que da las diversas direcciones de posiciones que utiliza el programa en código máquina. Ejecute este programa con la caja buffer y el convertidor D/A dispuestos tal como se explica en la página 1212. Si no posee ensamblador, entonces entre y ejecute este programa cargador en BASIC antes de ejecutar el programa de llamada. Si utiliza el cargador de BASIC, entonces puede omitir las líneas 45 y 50 del programa de llamada.

Los usuarios de BBC deben entrar la versión para el BBC tal como está escrita y ejecutarla.

```
 10 REM CARGADOR BASIC PARA PROGRAMA FREC/AMP
 20 :
 30 FOR I=49801 TO 49911
 40 READ A:POKEI,A
 50 CC=CC+A
 60 NEXT I
 70 READ CS:IF CC<>CS THENPRINT"ERROR SUMA CONTROL":STOP
100 DATA172,134,194,208,8,169,0,141
110 DATA204,194,76,174,194,169,15,141
120 DATA204,194,162,15,172,134,194,189
130 DATA0,194,24,106,157,15,194,136
140 DATA208,248,202,16,239,120,169,0
150 DATA141,133,194,174,135,194,189,30
160 DATA194,172,136,194,240,7,10,46
170 DATA133,194,136,208,249,141,132
180 DATA194,162,0,189,15,194,172,135
190 DATA194,136,208,253,141,1,221,232
200 DATA224,15,208,239,173,132,194,56
210 DATA233,1,141,132,194,173,133,194
220 DATA233,0,141,133,194,208,218,169
230 DATA0,205,132,194,208,211,88,96
240 DATA16186:REM * SUMA CONTROL *

 10 REM **** PROGRAMA DE LLAMADA ****
 20 REM ****                Y       ****
 30 REM ****  PREPARACION TABLA  ****
 40 :
 45 DN=8:REM SI CASSETTE DN=1
 50 IF A=0 THEN A=1:LOAD"FREC.HEXA",DN,1
 60 :
 70 REM **** PREPARACION TABLA FORMA ****
 75 :
 80 S=15:TB=12*4096+2*256
 90 FOR I=0 TO S-1
100 Y=127*SIN(X)+127
110 POKE TB+I, Y
120 X=X+2/S
130 NEXT I
140 :
150 REM **** PREPARACION TABLA FREC/DEMORA ****
160 :
170 TB=TB+2*S
180 FOR D=0 TO 101
190 TV=10^6/50*(271+75*D))
200 POKE TB+D,TV
210 NEXT D
```

```
; ++++++++++++++++++++
; ++++++++++++++++++++
; ++                  ++
; ++   GENERACION DE   ++
; ++    FRECUENCIA Y   ++
; ++      AMPLITUD     ++
; ++       CBM 64      ++
; ++                  ++
; ++++++++++++++++++++
; ++++++++++++++++++++

PASOS     =15          ;N.º DE PASOS POR CICLO DE ONDA
PUERTA    =56577
;
*=        $C200
SHAPTB    *=*+PASOS          ;TABLA FORMA ONDA
AMPTAB    *=*+PASOS          ;TABLA AMPLITUD
BUCLETB   *=*+102            ;TABLA FREC/DEMORA
CONTADOR  *=*+2              ;CONTADOR BUCLE
DIVISN    *=*+1              ;FACTOR DE DIVISION DE ONDA
DEMORA    *=*+1              ;FACTOR DEMORA
TIEMPO    *=*+1              ;FACTOR DURACION NOTA
;
;**** PREPARAR TABLA AMPLITUD ****
;
          LDY DIVISN
          BNE CONT
          LDA #SHAPTB        ;MODIFICAR PROGRAMA
          STA SIGVAL+1       ;PARA CARGAR SHAPTB
          JMP INIC
;
CONT
          LDA #<AMPTAB
          STA SIGVAL+1
          LDX #PASOS
NEXT
          LDY DIVISN
          LDA SHAPTB,X
MAS
          CLC
          ROR A
          STA AMPTAB,X
          DEY
          BNE MAS
          DEX
          BPL NEXT
;
;**** ESTABLECER VALOR CONTADOR ****
;
INIC
          SEI
          LDA #$00
          STA CONTADOR+1     ;INIC CONTADOR HIBYTE
          LDX DEMORA
          LDA BUCLETB,X
          LDY TIEMPO
          BEQ NOMULT
MULT
          ASL A
          ROL CONTADOR+1
          DEY
          BNE MULT
NOMULT
          STA CONTADOR
;
;**** BUCLE PRDGRAMA PRINCIPAL ****
;
PRAL
          LDX #$00
SIGVAL
          LDA AMPTAB,X
          LDY DEMORA
MASDEM                       ;BUCLE DEMORA
          DEY
          BNE MASDEM
;
          STA PUERTA
          INX
          CPX #PASOS
          BEN SIGVAL
;
;**** DECREMENTAR CONTADOR ****
;
          LDA CONTADOR
          SEC
          SBC #$01
          STA CONTADOR
          LDA CONTADOR+1
          SBC #$00
          STA CONTADOR+1
          BNE PRAL
          LDA #$00
          CMP CONTADOR
          BNE PRAL
          CLI
          RTS
```

```
 15 REM ************************************
 20 REM **                                **
 25 REM **          GENERACION DE         **
 30 REM **           FRECUENCIA Y         **
 40 REM **           AMPLITUD BBC         **
 50 REM **                                **
 60 REM ************************************
 90 MODE 7
 95 pasos=15:puerta=&FE60
 97 rdd=&FE62:? rdd=255:REM TODAS SALIDA
100 HIMEM=HIMEM-&101:REM RESERVAR ESPACIO TABLA
110 tabla_forma=HIMEM+1
112 tabla_amplitud=tabla_forma+pasos
114 tabla_bucle=tabla_amplitud+pasos
115 PROCpreparar_tablas
120 PROCcodigo_assemble
140 REM **** PROGRAMA DE PRUEBA BASIC ****
160 CLS
170 PRINT:INPUT"FACTOR DE AMPLITUD 0-7";FA
180 IF FA<0 OR FA>7 THEN 170
190 ?factor_div=FA
200 PRINT:INPUT"FACTOR DE DEMORA 1-101";FD
210 IF FD<1 OR FD>101 THEN 200
220 ?factor_demora=FD
230 PRINT:INPUT"FACTOR DE DURACION 0-15";FT
240 IF FT<0 OR FT>15 THEN 230
250 ?factor_tiempo=FT
265 REPEAT
270 CALL frec
280 A$=GET$
290 UNTIL A$="X"
300 GOTO 160:REM RECOMENZAR
900 END
999 :
1000 DEF PROCcodigo_assemble
1005 DIM MC% &FF
1010 FOR opt%=0 TO 3 STEP 3
1020 P%=MC%
1030 contador=P%:P%=P%+2
1040 factor_div=P%:P%=P%+1
1050 factor_demora=P%:P%=P%+1
1060 factor_tiempo=P%:P%=P%+1
1070 [
1075 OPT opt%
```

```
1080 \**** PREPARAR TABLA AMPLITUD ****
1090
1095 .frec
1100          LDY factor_div
1110          BNE cont
1120          LDA #tabla_forma MOD 256
1130          STA sigval+1
1140          JMP inic
1150\
1160 .cont
1170          LDA #tabla_amplitud MOD 256
1180          STA sigval+1
1190          LDX #pasos
1195 .next
1200          LDY factor_div
1210          LDA tabla_forma,X
1220 .mas
1230          CLC
1240          ROR A
1250          STA tabla_amplitud
1260          DEY
1270          BNE mas
1280          DEX
1290          BPL next
1300 \
1310 \****ESTABLECER VALOR CONTADOR****
1320 \
1330 .inic
1340          SEI
1350          LDA #0
1360          STA contador+1
1370          LDX factor_demora
1380          LDA tabla_bucle,X
1390          LDY factor_tiempo
1400          BEQ nomult
1410 .mult
1420          ASL A
1430          ROL contador+1
1440          DEY
1450          BNE mult
1460 .nomult
1470          STA contador
1480
1490  **** BUCLE PROGRAMA PRINCIPAL ****
1500
1510 .pral
1520          LDX #0
1530 .sigval
1540          LDA tabla_amplitud,X
1550          LDY factor_demora
1560 .masdem
1570          DEY
1580          BNE masdem
1590\
1600          STA puerta
1610          INX
1620          CPX #pasos
1630          BNE sigval
1640 \
1650 \****DECREMENTAR CONTADOR****
1660 \
1670          LDA contador
1680          SEC
1690          SBC #1
1700          STA contador
1710          LDA contador+1
1720          SBC #0
1730          STA contador+1
1740          BNE pral
1750          LDA #0
1760          CMP contador
1770          BNE pral
1780          CLI
1790          RTS
1800 ]
1810 NEXT opt%
1820 ENDPROC
1999 :
2000 DEF PROCpreparar_tablas
2005 x=0
2010 FOR I=tabla_forma TO tabla_forma+pasos-1
2020 y=127*SIN(x)+127
2030 ? I=y
2040 x=x+2*PI/pasos
2050 NEXT I
2060 :
2070 FOR demora=0 TO 101
2080 val_bucle=10^6/50*(271+75*demora))
2090 tabla_bucle?demora=val_bucle
2100 NEXT demora
2120 ENDPROC
```

Dispositivo «mágico»

He aquí un dispositivo que permite producir una imagen en pantalla con sólo orientar una cámara hacia el objeto

Ian McKinnell

Imágenes móviles EV1
El software de la cámara Snap se denomina EV1. Visualiza continuamente en la pantalla lo que la cámara «ve». Asimismo, permite volcar imágenes a una impresora, congelarlas en pantalla o guardarlas en disco. El ordenador calcula la exposición adecuada, si bien ésta se puede modificar de forma manual pulsando las teclas de flechas hacia arriba y hacia abajo

El EVI Video System, también conocido como "Snap" (instantánea), tiene un precio que se ajusta muy bien al presupuesto del usuario de un ordenador personal. Completo, con el software, cuesta menos de la mitad que cualquier interface para video comparable. El sistema Snap se compone de una pequeña cámara electrónica que se conecta mediante un cable plano a la puerta para el usuario del BBC, junto con software en cassette o bien en disco. Con este sistema se puede transferir la imagen de cualquier objeto o escena a la pantalla de su ordenador simplemente impartiendo la instrucción adecuada. El Snap se puede usar como mera diversión, pero también son posibles aplicaciones más serias: el sistema podría ser la base de una alarma antirrobo "inteligente", o se podría utilizar para reconocimiento de imágenes, quizá incluso confiriéndole "visión" a un robot.

El Snap opera en virtud de un truco de la electrónica. En el interior de la cámara, detrás de la lente, hay un chip de memoria RAM de 32 K. A diferencia de los chips normales, éste se ha fabricado con una ventana transparente en su superficie superior. La imagen de la lente se centra en la superficie del dispositivo de silicio que conforma el chip, en el cual hay una matriz de 256 × 128 diminutas celdas de memoria. Estas celdas, al igual que las de cualquier otro chip, poseen una propiedad que suele ser "invisible" para el usuario. Cuando una celda se expone a la luz, pierde lentamente su carga almacenada, y la velocidad a la cual se descarga es proporcional a la intensidad de la luz que recibe. En la cámara Snap, todas las celdas se cargan primero completamente escribiendo en todas las posiciones de memoria un valor específico, "encendiéndolas" (poniéndolas todas a ON). Al cabo de un breve período, las celdas que reciben luz se descargan lentamente. Después de una pausa, se vuelven a leer para evaluar el valor de cada una. Aquellas que no hayan recibido luz tendrán todavía el mismo valor "encendido", mientras que aquellas que hayan estado expuestas a suficiente luz se habrán descargado, modificando su valor almacenado, y, en consecuencia, se leerán como "apagadas". El software del sistema traza un punto iluminado en la pantalla en una posición que corresponde a cada celda "apagada" de la matriz de memoria. De este modo, se transfiere a la pantalla del BBC una reproducción de la imagen del chip.

El período de tiempo que transcurre entre que el ordenador escribe todas las celdas de memoria para "encenderlas" y las vuelve a leer, determina la "exposición" de la imagen. La cámara Snap tiene capacidad para producir muchas imágenes por segundo si la eliminación resulta suficientemente brillante: con la iluminación normal de una habitación se puede tomar una imagen en menos de un segundo.

La resolución de la imagen está limitada por la matriz de celdas de memoria de 256 × 128 del chip de la cámara. La misma no es particularmente alta (es aproximadamente la misma que la pantalla del BBC en Mode 2 o Mode 5), pero sí proporciona una calidad de imagen muy razonable, comparable a la de la fotografía de un periódico. Una restricción más importante en cuanto a la calidad no es la resolución sino que es consecuencia de otra característica del chip de memoria. La matriz de celdas está, en realidad, dividida en dos segmentos separados en la superficie del chip. Un agujero entre estos dos bloques significa que hay una franja a través del centro de la imagen que la cámara no detecta, y, por lo tanto, la imagen resultante en pantalla tiene una pequeña sección que falta. Sorprendente-

mente, esto no es demasiado grave y en muchas imágenes pasa totalmente desapercibido.

La cámara Snap se encuentra alojada en una pequeña caja plástica cuyo tamaño es similar al de un paquete de cigarrillos. En su parte frontal está la lente y en la base hay una montura para trípode estándar, que permite estabilizar la unidad cuando se necesitan exposiciones prolongadas. Dos metros de cable plano de ocho vías conectan la unidad con la puerta para el usuario del BBC. Éste es todo el hardware que se requiere.

La cámara utiliza una lente de la pequeña cámara reflex Pentax 110 de una sola lente, instalada en una montura estándar de tipo bayoneta. Tales lentes se pueden adquirir con toda facilidad en tiendas de artículos fotográficos y se venden en una amplia variedad de longitudes focales (se pueden instalar hasta lentes *zoom*), de modo que la lente que se suministra se puede sustituir fácilmente por otra. El único problema de esta disposición es que la lente del sistema Snap está colocada a una distancia del chip que es distinta de la distancia que guardaría, en la cámara Pentax, respecto a la película. Por consiguiente, las marcas de la distancia de enfoque de la lente no tienen ningún significado cuando se la utiliza con el sistema Snap.

La imagen en pantalla producida por la cámara depende en gran medida del software. Con el sistema se suministra un juego de programas y un manual de instrucciones de 50 páginas (preparado de forma muy profesional en un Apple Macintosh) que explica cómo se utiliza el software y los detalles relativos al funcionamiento de la máquina.

La cámara se puede utilizar de dos formas diferentes. Es posible producir imágenes a partir de una única imagen, lo que crea una visualización en pantalla de "dos tonos"; también se puede componer una imagen de "tonos múltiples" tomando varias imágenes con exposiciones distintas. El primer método lo emplea el primer programa del software que se suministra; éste utiliza una pequeña sección de pantalla en Mode 4 para presentar una imagen constantemente actualizada. La exposición se regula pulsando dos de las teclas para el cursor. Este programa incluye, asimismo, una rutina de vuelco de pantalla apta para impresoras Epson o compatibles con la misma.

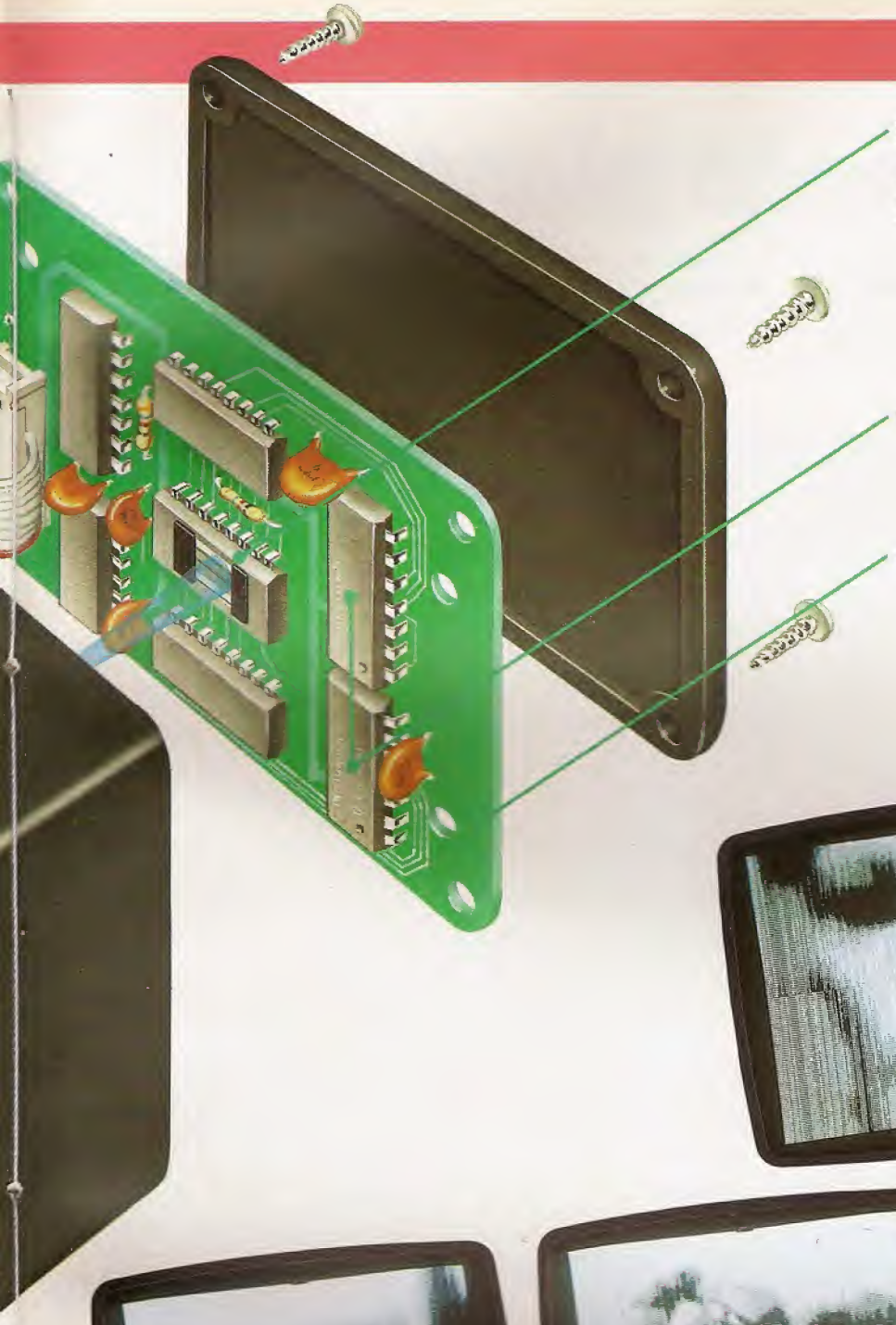
El segundo programa permite construir imágenes más realistas en varios tonos. Se toman ocho imágenes en distintas exposiciones, y éstas se visualizan simultáneamente en una pantalla de Mode 0, utilizando el sombreado para dar más énfasis a las imágenes de exposición corta. El efecto de esto es una imagen que contiene ocho grados distintos de brillo, de negro a blanco pasando por diversas tonalidades de gris. Esto da un efecto más realista, pero a la luz normal de una habitación la producción de estas imágenes en "escala de grises" puede llevar hasta 10 segundos: no del todo una "instantánea".

El programa *Secure* convierte su sistema Snap en una alarma antirrobo informatizada. El software toma nota sólo de las diferencias entre imágenes sucesivas, de manera que puede colocarse de forma que produzca una señal de alarma cuando estas diferencias alcancen un determinado nivel. Por ejemplo, si la cámara está orientada hacia el exterior de su casa, ignorará el hecho de que los árboles sean mecidos por el viento, pero disparará la alarma si llega un visitante a la puerta de entrada.

Steve Cross

Movie almacena una corta secuencia de fotogramas de dos tonos, que se reproducen luego rápidamente, creando un efecto de animación. También se suministra un programa llamado *Animal*. Éste es una versión en video del conocido juego por ordenador llamado *Animales* (véase p. 732). Para jugar a él el usuario debe presentar al sistema una imagen que contiene diversos objetos. El programa analiza la imagen, tomando nota de los perfiles de los distintos objetos que "ve", y luego invita al jugador a dar los nombres de cada uno. Si entonces el usuario dirige la cámara hacia otra imagen o escena, ésta intentará identificar cualquiera de los objetos que también hayan aparecido en la primera imagen.

Para los usuarios de ordenadores personales, tal vez la opción más atractiva sea el programa denominado *Arty*. Éste se utiliza para producir complejas imágenes en pantalla a partir de otras distintas. Se emplea la pantalla de Mode 1 completa y las imágenes captadas por la cámara se pueden posicionar en cualquier lugar de la pantalla, con tamaño ampliado o reducido, mediante el empleo de



SNAP/EV1 VIDEO
Dimensiones: 70 mm×50 mm×25 mm
Lentes: Lente de 24 mm o de 18 mm
Interfaces: Conector de 20 vías
Manuales: Guía para el usuario de la cámara Snap
Ventajas: Permite que el usuario guarde imágenes en pantalla o bien en disco
Desventajas: La resolución es pobre, lo que en ocasiones dificulta el control de la cámara

Resolución vertical
Las imágenes generadas por la cámara Snap se forman completamente a partir de líneas verticales, con espacios para reflejar las variaciones en el brillo. La resolución de las imágenes es aceptable, pero dista mucho de ser excepcional

Distintos puntos de palidez
Las imágenes de la cámara Snap normalmente se visualizan en blanco y negro. Un programa de escala de grises denominado *Grey* permite visualizar una imagen a través de toda la pantalla en ocho niveles de brillo. La acrecentada capacidad para reflejar contrastes de tonalidades aumenta el realismo de la imagen visualizada

una palanca de mando. Los colores de primer plano y fondo se pueden cambiar utilizando las teclas de función. Para aprovechar al máximo este programa se necesita cierta dosis de paciencia, pero a partir de unos cuantos objetos de la vida real se pueden construir complejas pantallas a todo color.

Si bien el software que se suministra es complejo y está bien elaborado, cubre una gama de aplicaciones limitada. El programa de imágenes en escala de grises, por ejemplo, sólo se puede emplear en Mode 0. Sería más útil una opción para hacerlo en Mode 2, usando como tonos de gris los distintos colores disponibles en esa modalidad. En el manual se ofrece una gran cantidad de información relativa a las rutinas en código máquina que utiliza el software, pero para hacer uso de las mismas se requiere conocer dicho lenguaje. El usuario de BASIC queda limitado al software suministrado.

Haciendo planes

Esta vez analizaremos el «Graph Plan», un paquete combinado de hoja electrónica y gráficos creado para el BBC Modelo B

A diferencia de los otros paquetes que hemos examinado en esta serie, *Graph Plan* es un programa basado en disco. Acorn regala el paquete, como parte de un conjunto de software gratuito, a los compradores del segundo procesador Z80 (necesario para el uso del paquete). Al igual que todos los programas de este lote, *Graph Plan* es un paquete muy útil y fiable.

El programa no tiene tanto "estilo" como el *Abacus* de Psion, el paquete de hoja electrónica/gráficos que reciben sin cargo alguno los usuarios del Sinclair QL (véase p. 1204). La ventaja sobre el *Abacus* reside en las rápidas velocidades de acceso y almacenamiento de que disfruta el software basado en disco. (Los microdrives del QL son adecuados si se está acostumbrado al software basado en cassette, pero resultan de una lentitud frustrante si uno está habituado a utilizar una verdadera unidad de disco.) Puesto que *Graph Plan* es un paquete "de regalo", es seguro que gozará de popularidad, aunque no tanto, tal vez, como el *Abacus*, porque el precio combinado del procesador Z80 más las unidades Acorn es muy superior al del QL.

En este capítulo concentraremos nuestra atención en la vertiente de gráficos de los modelos financieros. *Graph Plan*, como su nombre sugiere, posee unas capacidades para gráficos muy amplias, así como una formidable matriz de fórmulas comerciales y matemáticas incorporadas.

Lo que le confiere al *Graph Plan* su inusual estilo es su original forma de comunicarse mediante números: toda la interacción del usuario con el programa se realiza a través de 144 instrucciones numeradas. Al cargarlo, el programa tiene una visualización de hoja electrónica estándar (dividida en filas y columnas) que ocupa la mayor parte de la pantalla. En todo el lado derecho de ésta se visualizan las 20 instrucciones básicas con sus números. El usuario selecciona una instrucción y digita el número cuando así se le solicita en la tercera línea de estado, encima de la visualización, mediante el mensaje ENTER COMMAND (entre instrucción).

A pesar de que seleccionar una instrucción (ya sea desde el menú en pantalla o bien de la lista completa de instrucciones del excelente manual del *Graph Plan*, de 124 páginas) es suficientemente sencillo, esta forma de hacer las cosas tiene desventajas obvias. La mayor parte de los paquetes de modelos, en especial los que han obtenido tanto éxito, como el *Lotus 1-2-3* (véase p. 1124), exigen que el usuario entre solamente la letra inicial de una instrucción (o que la seleccione en una visualización mediante las flechas del cursor).

Los paquetes de modelos sofisticados, como el *Lotus 1-2-3*, visualizan explicaciones sobre la función de cada instrucción. *Graph Plan*, por el contrario, da por sentado que el usuario comprende la función de todas sus instrucciones. No obstante, el programa sí proporciona la facilidad para alterar la lista de instrucciones a la derecha de la pantalla, de modo que el usuario puede confeccionar la lista más útil para él. Si, por ejemplo, se selecciona la instrucción número 2 data, el menú de instrucciones pasa a visualizar las instrucciones de la 29 a la 48 (las instrucciones para entrada y manipulación de datos). Existe asimismo una facilidad HELP (ayuda) (instrucción número 7) que ofrece la explicación de la función de la instrucción dada.

Además del sistema de instrucciones numeradas, *Graph Plan* posee otras características exclusivas. La mayoría de las hojas electrónicas, por ejemplo, se basan en el concepto de "celda": una intersección entre una fila y una columna. Graph Plan trata las filas y las columnas como entidades separadas, y el "puntero de datos" es un indicador de cursor que, además de visualizar la identidad del cuadrado al que está apuntando el cursor, indica si se está en modalidad de fila o de columna.

Esta distinción carecería de significado en un sistema en el cual la dirección de celda individual fuera el punto central de referencia, pero en el *Graph Plan* reviste una enorme importancia, porque los gráficos se dibujan con referencia a filas o bien a columnas, pero no a ambas. Para indicar al paquete si se desea generar un gráfico basado en fila o basado en columna, debe establecerse la modalidad correcta cambiando el puntero de datos. Esto se realiza pulsando las teclas con flecha, para desplazar el cursor hasta el encabezamiento de una fila o de una columna, lo cual automáticamente establece la modalidad correspondiente.

Un modelo sencillo

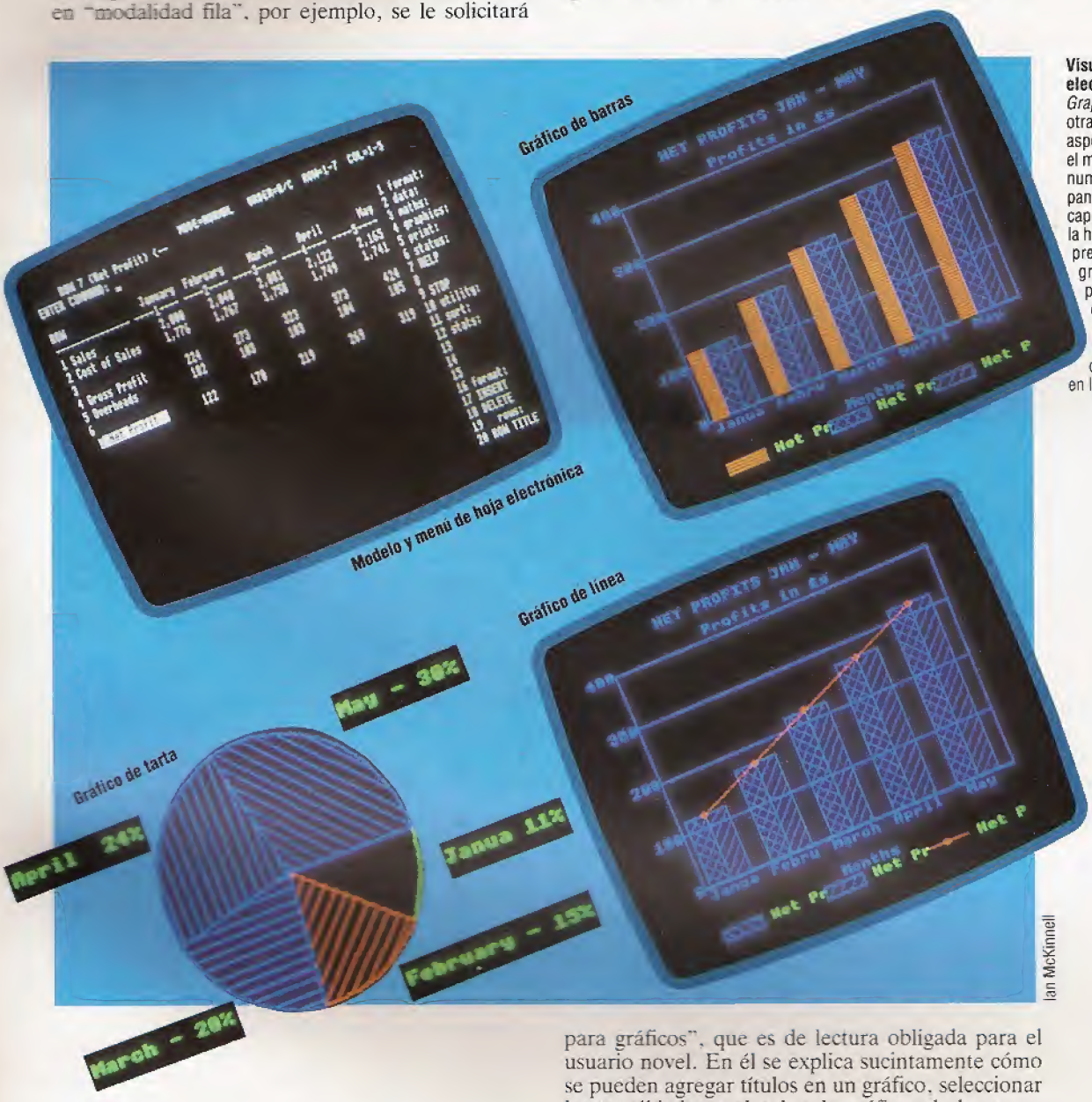
A modo de ilustración de la técnica para dibujar gráficas del paquete, vamos a considerar un modelo sencillo. Posee cinco columnas, rotuladas de "Enero" a "Mayo", respectivamente, y cinco filas, denominadas "Ventas", "Costes de ventas", "Beneficios brutos", "Gastos generales" y "Beneficios netos". En un modelo como éste, una gráfica basada en filas tendrá un significado diferente al de una gráfica basada en columnas. Por ejemplo, podríamos generar un gráfico de barras muy sencillo basado en filas para las cifras del movimiento total de ventas para los meses de enero a mayo.

El gráfico visualizaría los títulos de las columnas (de "Enero" a "Mayo") a lo largo del eje *x*, y las barras representarían los valores dados en la fila uno del modelo. Por el contrario, situando el puntero de datos en la modalidad de columna, podríamos generar una gráfica de barras muy distinta. Ésta visualizaría los títulos de las filas ("Ventas", "Coste de ventas", etc.) a lo largo del eje *x*, con las barras representando los valores dados en la columna uno ("Enero") del modelo.

Con los datos apropiados, *Graph Plan* puede producir instantáneamente una cantidad considerable de gráficos diferentes a partir de los datos suministrados en este modelo. Todos estos gráficos se pueden ver, a su vez, en la pantalla, sin ninguna intervención por parte del usuario. *Graph Plan* permite hacer esto mediante la utilización de la instrucción de gráficos 62 SELECT. Entrando 62 en respuesta al mensaje ENTER COMMAND se genera otro mensaje en la línea de estado tres. Si usted se halla en "modalidad fila", por ejemplo, se le solicitará que dé el número de la fila de datos que desea representar gráficamente. Apenas acaba de seleccionar la fila, el mensaje cambia y pasa a solicitarle qué tipo de gráfico desea: "Choose (Bar=1, Line=2, Pie=3)" (Elija: De barras=1, De línea=2, De tarta=3). Entonces puede contemplar inmediatamente el gráfico seleccionando la instrucción 61 DISPLAY. Y puede pasar de gráficos de barras a gráficos de líneas o en forma de tarta a voluntad.

Si desea más flexibilidad en el trazado y el diseño de un gráfico, puede emplear la instrucción 63 OPTIONS. El manual visualiza un "diagrama de decisión" muy bueno, que ilustra claramente el proceso de selección que ha de seguirse cuando se utiliza esta instrucción. La instrucción 63 le presenta en pantalla un menú de seis opciones: Display Chart, Define Chart Options, Define Axes Options, Define Pie Options, Print Chart y Plot Chart (visualizar diagrama, definir opciones del diagrama, definir opciones de eje, definir opciones de tarta, imprimir diagrama y trazar diagrama). Además, el manual posee un apéndice especial, titulado "Guía de los submenús para gráficos", que es de lectura obligada para el usuario novel. En él se explica sucintamente cómo se pueden agregar títulos en un gráfico, seleccionar las tonalidades o el color de gráficos de barras, y realizar toda clase de variaciones de escala (hasta hacer los ejes logarítmicos en vez de lineales).

Una buena facilidad para gráficos incorporada en forma de hoja electrónica (que posee funciones matemáticas y estadísticas) hace que el *Graph Plan* sea apto para una amplia gama de aplicaciones científicas y de ingeniería relativamente sencillas. El paquete proporciona un medio excelente para presentar datos, ya sea para informes o para conferencias, y, por consiguiente, les agradará tanto a los técnicos y a los científicos como a los usuarios comerciales.



Ian McKinnell

Visualización de hoja electrónica
Graph Plan se diferencia de las otras hojas electrónicas en dos aspectos. El primero de ellos es el menú de instrucciones numeradas, a la derecha de la pantalla. El segundo es la capacidad de tomar datos de la hoja electrónica y presentarlos en forma gráfica. Graph Plan puede visualizar datos de una fila o de una columna en tres formatos diferentes, como vemos en las fotografías

Poesía en movimiento

En este capítulo centraremos nuestra atención en el tratamiento de listas, fundamental para la forma en que opera este lenguaje

Una lista es un conjunto ordenado de objetos, y en LOGO se identifica mediante la utilización de corchetes; de modo que [CEILAN MADRAS VINDALOO] es una lista. En esta serie ya nos hemos encontrado en varias ocasiones con listas. De hecho, en LOGO no podemos prescindir de ellas, porque es un lenguaje basado en listas. Ya hemos visto cómo una definición para un cuadrado (REPEAT 4 [FD 50 RT 90]) posee una lista de instrucciones (encerradas entre corchetes) como su segunda entrada. Del mismo modo, MAKE "ENT REQUEST le asigna a ENT una lista compuesta por la entrada desde el teclado.

Se pueden asignar listas a variables locales: por ejemplo, MAKE "CURRY [CEILAN MADRAS VINDALOO]. La instrucción PRINT :CURRY imprime la lista sin los corchetes: CEILAN MADRAS VINDALOO.

En LOGO, un objeto puede ser un número, una palabra o una lista, y esta última se define simplemente como un conjunto de objetos. Ésta, por supuesto, es una definición recursiva; una lista puede contener otra lista, o una lista de listas, y así sucesivamente. [[POLLO TIKKA] NAN ENSALADA] es una lista válida, teniendo como primer elemento una lista ([POLLO TIKKA]). Los procedimientos recursivos suelen ser necesarios para procesar listas, precisamente porque éstas son objetos recursivos.

La mayor parte de nuestra programación en LOGO ahora se ha referido a un número o una palabra cada vez. Cuando deseamos procesar grupos de objetos simultáneamente, necesitamos organizar estos objetos simples en una única unidad. El LOGO tiene en la lista su método básico para agrupar objetos simples. La elección de ésta se debe a que es muy versátil: partiendo de una lista se puede crear cualquier organización compleja de datos.

Las dos operaciones de lista fundamentales son FIRST y BUTFIRST. FIRST [CEILAN MADRAS VINDALOO] produce CEILAN, es decir, nos da el primer elemento de la lista. BUTFIRST [CEILAN MADRAS VINDALOO] produce MADRAS VINDALOO; en otras palabras, nos la proporciona sin su primer elemento.

He aquí un procedimiento que imprime los elementos de la lista, uno debajo del otro:

```
TO IMPRIMIR :LISTA
  PRINT FIRST :LISTA
  IMPRIMIR BUTFIRST :LISTA
END
```

De modo que IMPRIMIR [CEILAN MADRAS VINDALOO] nos da:

```
CEILAN
MADRAS
VINDALOO
```

La primera instrucción imprime el primer elemento y después pasa la tarea de imprimir el resto de la lista de entrada a otra copia del procedimiento IMPRIMIR. Cuando ejecute este procedimiento, al acabarse los datos obtendrá un mensaje de error. Ésta es una forma más elegante de terminar:

```
TO IMPRIMIR :LISTA
  IF EMPTY? :LISTA THEN STOP
  PRINT FIRST :LISTA
  IMPRIMIR BUTFIRST :LISTA
END
```

EMPTY? (¿vacía?) verifica si su entrada es la "lista vacía": []. Algunas versiones MIT no poseen la primitiva EMPTY?, pero ésta puede definirse así:

```
TO EMPTY? :LISTA
  IF :LISTA = []THEN OUTPUT "TRUE
  OUTPUT "FALSE
END
```

Similares a FIRST y BUTFIRST son LAST y BUTLAST. LAST [CEILAN MADRAS VINDALOO] produce VINDALOO, y BUTLAST [CEILAN MADRAS VINDALOO] produce CEILAN MADRAS.

Para nuestra primera exploración del tratamiento de listas, intentaremos imitar algunos balbuceos al azar en el diván del psicoanalista. Primero le asignaremos a la variable PALABRAS todos los vocablos que conocemos:

```
MAKE "PALABRAS [MADRE PADRE SEXO
ASESINATO CELOS FUEGO MAR MUERTE SUEÑO]
```

Queremos producir un flujo constante y al azar de estas palabras, porque la experiencia nos ha demostrado que son éstos los vocablos que siempre resultan útiles para captar la atención del psicoanalista.

Para obtener un elemento aleatorio necesitamos seleccionar un número al azar, n, entre uno y la longitud de la lista (nueve, en este caso) y luego seleccionar el enésimo elemento de ésta.

```
TO ENE :NO :LISTA
  IF :NO=1 THEN OUTPUT FIRST :LISTA
  OUTPUT ENE :NO-1 BUTFIRST :LISTA
END
```

Vamos a utilizar este procedimiento en algunos ejemplos, para ver cómo funciona. Supongamos que usted digita ENE 1 :PALABRAS. La condición de la primera línea es verdadera, de modo que el procedimiento produce FIRST :PALABRAS, que en nuestro ejemplo es MADRE.

Pruebe con ENE 2 :PALABRAS: ahora la condición es falsa, de modo que el procedimiento produce ENE 1 BUTFIRST :PALABRAS. Éste ignora el primer elemento de la lista y toma la primera palabra del resto de la misma: PADRE.

De manera que nuestro procedimiento para imprimir al azar una palabra de nuestro limitado vocabulario sería:

```
TO TOMARALAZAR :LISTA
  OUTPUT ENE ((RANDOM 9)+1) :LISTA
END
```

Para utilizarlo, digite TOMARALAZAR :PALABRAS.

Nuestro procedimiento está restringido a listas de nueve elementos. Podríamos mejorarlo si pudiéramos determinar cuántos hay en una lista determinada. Éste es un procedimiento que realiza eso:

```
TO LONGITUD :LISTA
  IF EMPTY? :LISTA THEN OUTPUT 0
  OUTPUT 1 + LONGITUD BUTFIRST :LISTA
END
```

Para ver cómo funciona, pruebe con: LONGITUD [CIENCIA FICCION]. Como la lista contiene algunas palabras, la condición fracasa, de modo que el procedimiento produce 1 + LONGITUD [FICCION]. Ahora LONGITUD [FICCION] produce 1 + LONGITUD []. Al llamar a LONGITUD con una entrada de [], la condición de la línea 1 es verdadera, por lo que el procedimiento produce 0. Ahora LONGITUD [FICCION] produce 0 + 1 = 1 y finalmente LONGITUD [CIENCIA FICCION] produce 1 + 1 = 2. Por lo tanto, un procedimiento más general para tomar palabras al azar de una lista de cualquier longitud sería:

```
TO TOMARALAZAR :LISTA
  OUTPUT ENE ((RANDOM LONGITUD :LISTA)
  +1) :LISTA
END
```

Muchas versiones de LOGO poseen una primitiva, ITEM, que hace exactamente lo que hace ENE, y una primitiva denominada COUNT que realiza lo mismo que LONGITUD. Utilizando estas primitivas, podemos reescribir el procedimiento:

```
TO TOMARALAZAR :LISTA
  OUTPUT ITEM ((RANDOM COUNT :LISTA)
  +1) :LISTA
END
```

Para imprimir una selección de 10 términos que hagan que su psicoanalista se mantenga atento, simplemente digite:

```
REPEAT 10 [PRINT TOMARALAZAR :PALABRAS]
```

Existe un patrón para estos programas procesadores de listas que han compartido muchos de nuestros procedimientos repetitivos para gráficos tortuga. Éste patrón es:

- Si la tarea a realizar es muy sencilla, entonces hágala y deténgase.
- De lo contrario, efectúe una pequeña parte de la tarea.
- Después pase el resto de la tarea a otro procedimiento (con frecuencia una copia del procedimiento original).

Existe una estrategia que ofrece elevados porcentajes de éxito, con la cual nos encontraremos reiteradamente en los programas de tratamiento de listas. Compare este programa para dibujar polígonos:

```
TO POLI :N
  IF :N = 0 THEN STOP
  FD 30 RT (360/:N)
  POLI :N - 1
END
```

con la versión de IMPRIMIR que ofrecimos anteriormente. La estructura de ambos procedimientos es idéntica.

Poesía aleatoria

Después de no haber logrado impresionar a nuestro psicoanalista, ahora dedicamos nuestro esfuerzo a la poesía. Aquí produciremos frases completas en lugar de palabras individuales.

```
TO POEMA1 :LONGITUD
  IF :LONGITUD = 0 THEN PRINT "STOP
  (PRINT1 "' ' TOMARALAZAR :PALABRAS)
  POEMA 1 :LONGITUD - 1
END
```

PRINT1 "' ' se incluye para imprimir un espacio entre cada palabra. Para utilizar este procedimiento digite POEMA1 6 para una frase de seis vocablos.

Sería útil poder ampliar nuestra lista original de palabras sin tener que tomarnos el trabajo de volverla a escribir entera. Una manera de lograrlo es a

Abreviaturas

BUTFIRST	BF
BUTLAST	BL
SENTENCE	SE

Sir John Tenniel

Canta la Tortuga Artificial
He aquí un extracto de la canción de la Tortuga Artificial, personaje del famoso cuento *Alicia en el País de las Maravillas*, de Lewis Carroll. El patrón de la métrica, un poco difícil de obtener en una poesía generada por ordenador, está adaptado de una antigua canción popular de origen negro

"¿Por qué no vas más aprisa?", le dijo una pescadilla
a un caracol.
"Tras nosotros viene, muy cerca, un delfín pisándome
la cola.
¡Fíjate cuán raudas las langostas y las tortugas
avanzan todas!
Están esperándonos sobre el cascajo. ¿No querrás venir
y bailar también?"
Querrás, querrías, querrás, querrías,
¿no querrás tú bailar también?
Querrás, querrías, querrás, querrías,
¿no querrías tú también bailar?

"No sabes, no puedes saber, cuán agradable es el
vaivén
cuando levantándonos nos arrojen con las langostas
¡hacia el mar!"
Pero el caracol respondía: "¡Muy lejos! ¡Demasiado lejos!",
y ni se dignaba mirar a dónde.
Dijo que le agradecía a la pescadilla la invitación,
pero que al baile no se uniría.
No querría, no podría, no querría, no podría,
no querría bailar también.
No querría, no podría, no querría, no podría,
no podría también bailar.

través de la utilización de la operación SENTENCE (frase), que toma dos entradas y elabora con ellas una lista. De modo que SENTENCE "MERMELADA [JARRA MIEL] produce [MERMELADA JARRA MIEL].

```
TO AGREGARPALABRAS1 :LISTA
  MAKE "PALABRAS SENTENCE :LISTA :PALABRAS
END
```

De manera que ahora podemos ampliar PALABRAS mediante AGREGARPALABRAS [ANSIEDAD REPRESION [MIEDO A VOLAR]]. Pero se nos presenta el problema de que a la variable PALABRAS no se le ha asignado previamente ningún valor. Para obviar este contratiempo se utiliza la primitiva THING?, que verifica si a una variable se le ha asignado algún valor; produce verdadero (*true*) si su entrada tiene algún valor asignado. Ahora podemos incrementar nuestra lista mediante AGREGARPALABRAS1:

```
TO AGREGARPALABRAS1 :LISTA
  IF NOT THING? "PALABRAS THEN MAKE
  "PALABRAS []
  MAKE "PALABRAS SENTENCE :LISTA :PALABRAS
END
```

Empleando una lista de palabras diferente, obtuvimos la siguiente "poesía" aplicando este procedimiento:

```
APARICION RUIDOSAMENTE HABLO ESPLENDIDO
PARANOICO PLANETA ATERRORIZADO LA CON
VERDE APARICION FLOTANDO PARANOICO ROBOT
HOMBRE VOLO HABLABA FLOTANDO
RUIDOSAMENTE
```

Uno de los defectos más evidentes de nuestra poesía informatizada es su ignorancia total de la gramática. Los poemas podrían tener más sentido si pudiésemos ceñirnos a algunos patrones sintácticos simples, tales como: sustantivo, verbo, sustantivo. Una forma de hacer esto consiste en tener varias listas, una para cada parte de la oración. Entonces podríamos elegir una palabra de cada lista según la estructura que deseáramos para la frase.

Vamos a dejar este problema para que lo investigue usted mismo. En el próximo capítulo del curso le enseñaremos algunas maneras de mejorar las aptitudes de la tortuga para escribir poesía.

Complementos al LOGO

Algunas versiones de LOGO MIT no poseen EMPTY?, ITEM ni COUNT. En todas las versiones LCSI utilice:

EMPTYP por EMPTY?
LISTP por LIST?
TYPE por PRINT1

Hay una primitiva, EQUALP, que verifica si sus dos entradas son la misma. Empléela para comparar listas y palabras en lugar del signo (=). (El signo de igualdad funciona para listas en algunas versiones LCSI, pero en otras no.)

Recuerde la sintaxis diferente de IF:
IF EMPTYP :LISTA [OUTPUT 0]
En el LOGO Atari utilice SE por SENTENCE, y tenga en cuenta que ITEM no está implementado

Ejercicios

1) Escriba un procedimiento para imprimir una lista por orden inverso (utilice LAST y BUTLAST). Modifíquelo para que produzca la lista invertida
2) Escriba un procedimiento que elimine un elemento de una lista. DELETE "COMIDA [BEBIDA COMIDA] produce [BEBIDA] y DELETE "VINO [BEBIDA COMIDA] produce [BEBIDA COMIDA]

Respuestas a los ejercicios

Respuestas a los ejercicios de p. 1217:

1. Cálculo de potencias:

```
TO POTENCIA :A :N
  IF NOT ((INTEGER :N)=:N) THEN PRINT [SOLO
  INDICES DE NUMEROS ENTEROS] STOP
  IF :N=0 THEN OUTPUT 1
  OUTPUT :A* POTENCIA :A :N – 1
END
```

2. Conversión a hexadecimal:

```
TO IMPRESION.HEXA :N
  IF :N < 10 THEN OUTPUT :N
  IF :N = 10 THEN OUTPUT "A
  IF :N = 11 THEN OUTPUT "B
  IF :N = 12 THEN OUTPUT "C
  IF :N = 13 THEN OUTPUT "D
  IF :N = 14 THEN OUTPUT "E
  IF :N = 15 THEN OUTPUT "F
END

TO HEXA :N
  IF :N = 0 THEN STOP
  HEXA QUOTIENT :N 16
  PRINT1 IMPRESION.HEXA REMAINDER :N 16
END
```

3. Comprobar si un número es par:

```
TO PAR? :N
  IF ((REMAINDER :N 2)=0)THEN OUTPUT
  "TRUE OUTPUT "FALSE
END
```

4. Hallar una superficie utilizando el método Monte Carlo:

```
TO MC
  DRAW PU MAKE "IN 0
  MC1 1000 10 100
  (PRINT [LA SUPERFICIE ES] (:IN))
END

TO MC1 :N :XN :YN
  IF :N=0 THEN STOP
  PUNTO.ALEATORIO :XN :YN
  IF DENTRO? THEN MAKE "IN :IN+1
  MC1 :N – 1 :XN :YN
END

TO PUNTO.ALEATORIO :XN :YN
  SETXY RANDOM :XN RANDOM :YN
END

TO DENTRO?
  IF YCOR< XCOR*XCOR THEN OUTPUT
  "TRUE OUTPUT "FALSE
END
```

Squash

¿Por qué no hacer un poco de deporte ante su pequeña pantalla? Esta versión de «Squash» ha sido escrita para el microordenador Thomson TO 7

Gracias a su ordenador, puede jugar a squash cómodamente sentado en un sillón. La raqueta se desplaza con la ayuda de la palanca de mando o de las teclas Q, S y la barra espaciadora. Dispone de diez pelotas que ha de mantener en juego el mayor tiempo posible. Cada pelota que se devuelve proporciona un punto.

```
 10 REM **********
 20 REM * SQUASH *
 30 REM **********
 40 CLEAR ,,2
 50 GOSUB 660
 60 GOTO 180
 70 D=2* ((STICK (0)=7)-(STICK (0)=3))
 80 IF D<>0 THEN DO=D
 90 IF STICK (0)=0 THEN DO=0
100 RX=RX+DO
110 RETURN
120 D$=INKEY$
130 D=2* ((D$="Q")-(D$="S"))
140 IF D<>0 THEN DO=D
150 IF D$=N$ THEN DO=0
160 RX=RX+DO
170 RETURN
180 LOCATE BX,BY
190 COLOR 3,6
200 PRINT N$;
210 BX=BX+DX
220 BY=BY+DY
230 LOCATE BX,BY
240 PRINT B$;
250 IF BY=22 AND ABS (BX-RX-3) >1 THEN 360
260 IF BY=22 THEN S=S+1:BEEP:DY=-DY
270 IF BY=1 THEN BEEP:DY=-DY
280 IF BX=2 OR BX=37 THEN BEEP:DX=-DX
290 ON JS GOSUB 70,120
300 IF RX<0 THEN RX=0
310 IF RX>33 THEN RX=33
320 LOCATE RX,RY
330 COLOR 2
340 PRINT R$;
350 GOTO 180
360 NB=NB+1
370 IF NB=11 THEN 480
380 LOCATE BX, BY
390 PRINT N$;
400 FOR I=1 TO 3
410 BEEP
420 FOR J=1 TO 100
430 NEXT J
440 NEXT I
450 DO=0
460 GOSUB 940
470 GOTO 180
480 LOCATE 13,5
490 COLOR 0
500 PRINT "PUNTUACION :";S;
510 IF S>R1 THEN R1=S
520 LOCATE 13,10
530 PRINT "RECORD :";R1;
540 LOCATE 13,15
550 PRINT "OTRA ?";
560 NB=0
570 S=0
580 D$=INKEY$
590 IF D$<>" " THEN 580
600 D$=INKEY$
610 IF D$=" " THEN 600
620 IF D$<>"N" THEN 50
630 SCREEN 4,6,6
640 CLS
650 END
660 CLS
670 SCREEN 1,6,6
680 ATTRB 1,1
690 DEFINT A-Z
700 DEFGR$ (0)=255,255,255,0,0,0,0,0
710 DEFGR$ (1)=24,126,126,255,255,126,126,24
720 N$=CHR$(32)
730 LOCATE 1,10,0
740 PRINT "JOYSTICK (S o N) ?";
750 D$=INKEY$
760 IF D$="" " THEN 750
770 IF D$="S" THEN JS=1 ELSE JS=2
780 CLS
790 ATTRB 0,0
800 COLOR 6,1
810 FOR BX=1 TO 38
820 LOCATE BX,0
830 PRINT N$;
840 NEXT BX
850 FOR BY=1 TO 22
860 LOCATE 1,BY
870 PRINT N$;
880 LOCATE 38,BY
890 PRINT N$;
900 NEXT BY
910 R$=N$+N$+GR$ (0)-GR$ (0)+GR$ (0)+N$+N$
920 B$=GR$ (1)
930 RX=16
940 RY=23
950 BY=22
960 BX=INT (RND*34)+3
970 DY=-1
980 DX=(INT (RND*2)-0.5)*2
990 RETURN
```

A prueba de error

Vamos a poner un ejemplo de diseño de arriba abajo («top-down»): un programa en assembly para depurar y poner a punto otros programas («debugger»)

Veamos primeramente las fases de especificación y diseño de que se compone la obtención de un eliminador de errores. La especificación es bastante inmediata; ya tuvimos ocasión de considerar las funciones que suponemos se incorporarán en el programa (véase p. 1219).

Inputs del programa:

1. Un programa a depurar: Supondremos que ya está cargado en la memoria, junto con el depurador.
2. Órdenes: Debemos decidir si las órdenes se introducirán directamente o por medio de opciones de menú. Entraremos órdenes de un solo carácter según la tabla al margen.
3. Direcciones: Como pueden ser introducidas en hexadecimal, habrá que convertir una cadena de dígitos hexadecimales ASCII en números binarios de 16 bits.

Outputs del programa:

1. "Ecos" de los caracteres entrados: Téngase en cuenta que por el hecho de apretar una tecla no se obtiene un carácter en la pantalla. El ordenador debe ser programado para que haga eso.
2. Números de 8 y 16 bits: Son aceptados en forma de cadenas de números hexadecimales.
3. Cadenas: Que sirvan como etiquetas de los números anteriores.

B	insertar Punto de Ruptura (*B*reakpoint)
U	eliminar Punto de Ruptura (*Un*-insert)
D	visualizar (*D*isplay) Puntos de Ruptura en curso
S	comenzar (*S*tart) ejecución programa
G	ir (*Go*). Reemprender ejecución programa
R	visualizar contenido *R*egistros
M	inspeccionar y cambiar posición de *M*emoria
Q	salir (*Q*uit)

Hay muchas formas de dividir un programa en módulos y éstos en subrutinas, pero siempre debe proporcionarse un módulo "externo" que sirva de caparazón (*shell*) o enlace de los restantes. En nuestro programa este módulo tomará la siguiente forma:

Módulo principal

Data:
Dirección de inicio del programa (16 bits)
Aviso para la entrada de órdenes (el carácter ASCII individual '>')
Carácter de Orden, está en ASCII y es individual (¿usaremos caracteres en minúscula?)
Dirección de Ruptura, es la dirección de la rutina administradora de la interrupción SWI

Proceso:
Establecer la Interrupción
GET (tomar) la Dirección de Inicio
REPEAT
  DISPLAY aviso
REPEAT
  Tomar la Orden
UNTIL que la Orden sea válida
DISPLAY la Orden (eco)
IF Orden = 'B' THEN
  Insertar Punto de Ruptura
ELSE IF Orden = 'U' THEN
  Eliminar Punto de Ruptura
ELSE IF...
  Hasta que Orden = 'Q'
Fin del módulo principal

Con esto ya se tiene una buena idea de las rutinas que harán falta. No es lo mismo un módulo que una rutina. Hay varias rutinas que pueden agruparse lógicamente por compartir los mismos datos: por ejemplo, un módulo en nuestro caso tratará los puntos de ruptura. La fase siguiente muestra cómo es posible diseñar este módulo:

Módulo de rupturas

Data:
Tabla-Puntos-Ruptura es una matriz de direcciones de 16 bits que almacenará las direcciones de los puntos de ruptura
Valores-Sustituidos es una matriz de valores de 8 bits relativos a la tabla anterior. Aquí pueden almacenarse los opcodes que son sustituidos por una instrucción SWI en el punto de ruptura
Número-Puntos-Ruptura es un valor de 8 bits que da el número de puntos de ruptura activos
Punto-Ruptura-Siguiente es un valor de 8 bits, que contiene el siguiente punto de ruptura según avanza la ejecución
SWI-Opcode es un opcode de 8 bits para la instrucción SWI

Proceso 1: Inserción Puntos-Ruptura
IF Número-Puntos-Ruptura < MAX THEN
  Tomar-Dirección
  Añadir 1 a Número-Puntos-Ruptura
  Almacenar Dirección en Tabla-Puntos-Ruptura (Número-Puntos-Ruptura)
ENDIF
Fin del proceso 1

Proceso 2: Establecer-Punto-Ruptura (N)
(N nos dirá cuál de los puntos de la tabla ha de ser activado)
  Tomar-Dirección en Tabla-Puntos-Ruptura (N)
  Tomar el Opcode contenido en esa Dirección
  Almacenarlo en Valores-Sustituidos (N)
  Almacenar el Opcode SWI en Dirección
Fin del Proceso 2

El Proceso 2 se encuentra en una fase en la que podríamos empezar a codificar. Cuatro son los valores de datos que han de ser manipulados: N, el parámetro indicador del punto de ruptura que hay que usar, es un número de 8 bits que emplearemos como desplazamiento en las dos tablas y que oscilará entre uno y Número-Puntos-Ruptura−1. Nóte-

se, sin embargo, que una tabla toma valores de 16 bits mientras la otra los toma de 8 bits. Asumiremos que N es pasado en A. La dirección del punto de ruptura tomada de dicha tabla se introducirá en X. El opcode sustituido será llevado a B para su introducción en la tabla Valores-Sustituidos. Así B puede ser empleado para colocar el opcode SWI en la dirección apropiada.

Proporcionamos aquí la codificación final del Proceso 2 (Módulo Establecer-Punto-Ruptura); la siguiente tarea consistirá en diseñar un módulo para manejar las entradas y salidas. De lo que llevamos dicho, se puede colegir que existen diferentes tareas de E/S a ejecutar por el programa. Por el momento, asumiremos la existencia de dos subrutinas: INCH, encargada de introducir un carácter individual en el registro A a partir del teclado; y OUTCH, que enviará el carácter desde A hasta la posición de la pantalla que indique el cursor. Este módulo requiere las siguientes subrutinas:

1. TomarOrden: Entra desde el teclado la orden siguiente.
2. TomarDirección: Toma del teclado una dirección hexadecimal de 1 a 4 caracteres de longitud.
3. TomarValor: Toma un valor hexadecimal de 1 o 2 caracteres de longitud para modificar el valor de una posición de memoria.
4. VisualizarValor: Visualiza en pantalla un valor hexadecimal de dos caracteres.
5. VisualizarDirección: Visualiza en pantalla una dirección hexadecimal de cuatro caracteres.

Nuestro enfoque muestra la diferencia entre los métodos de programación de arriba abajo (*top-down*) y de abajo arriba (*bottom-up*). El primero puede llevarnos a definir y codificar estas operaciones de manera independiente, acabando por escribir rutinas separadas que esencialmente hacen la misma cosa.

El método ascendente puede ahorrarnos tiempo, espacio y esfuerzo, al escribir sencillamente unas cuantas rutinas útiles que emplearemos en distintas circunstancias.

Estas rutinas son:

GETCH: Entrar un carácter individual en A, comprobándolo en una lista de caracteres válidos (letras de órdenes o dígitos hexa), haciendo el eco de los caracteres válidos e ignorando los restantes.
GETHX2: Sirve para que GETCH tome dos dígitos hexa y los convierta en un número de ocho bits.
GETHX4: Toma cuatro dígitos hexa y forma un número de 16 bits.
PUTHEX: Visualiza un número de 8 bits como dos dígitos hexa (puede ser llamada dos veces para que visualice un número de 16 bits).
PUTCR: Para sacar un carácter de retorno de carro (o retorno de carro y salto de línea si es necesario).

Cada una de estas rutinas han de ser desarrolladas. Consideremos primero el diseño de GETCH.

La rutina GETCH

Data:
Car-In es un carácter ASCII entrado desde teclado (contenido en A)
Car-Válidos retiene la dirección de 16 bits de la tabla de caracteres válidos
Número-Car-Val es un valor de 8 bits
Car-Inspec es un contador de 8 bits

Proceso:
REPEAT
  Tomar el siguiente Car-In
  Poner Car-Inspec a (Número-Car-Val-1)
  While Car-Válidos (Car-Inspec) <> Car-In
    AND Car-Inspec > =0
    Decrementar Car-Inspec
Until Car-Inspec > = 0
DISPLAY Car-In

Para codificar esto, debemos emplear A para almacenar Car-In, y el valor de 16 bits de Car-Válidos puede ser pasado y guardado en X. En B puede ser pasado Número-Car-Val, pero necesita una mayor permanencia, lo que conseguiremos llevándolo a la pila. En este caso podemos emplear B para Car-Inspec. Observe que B devolverá el desplazamiento a la tabla, lo que será útil para interpretar las órdenes y para la conversión hexadecimal.

He aquí la forma final de esta rutina codificada. En la siguiente lección desarrollaremos las restantes rutinas necesarias para el módulo de entrada-salida.

Rutina GETCH

```
GETCH   PSHS    B       Salva B
REPTOO  BSR     INCH    Toma el siguiente Car-In
        LDB     1,S     Activa Car-Inspec
        DECB            Resta 1 del desplaz. máximo
WHILOO  BLT     ENDWOO  Mientras Car-Inspec>=0
        CMPA    B,X     AND
        BEQ     ENDWOO  Car-In<>Car-Válidos(Car-
                        Inspec)
        DECB            Decrementa Car-Inspec
        BRA     WHILOO
ENDWOO  TSTB
UNTLOO  BLT     REPTOO  Hasta que Car-Inspec>=0
        BSR     OUTCH   Visualiza Car-In
        LEAS    1,S     Incrementa S para "olvidar"
        RTS             el valor primitivo de B
```

Módulo Estab.-Punto-Ruptura

```
                        Declaraciones de datos
TABPR   RMB     32          Tabla-Puntos-Ruptura
TABVS   RMB     16          Valores-Sustituidos
NUMPR   FCB     0           Número-Puntos-Ruptura
NEXTPR  FCB     0           Siguiente-Punto-Ruptura
OPSWI   FCB     $3F         Opcode-SWI
MAXPR   FCB     16          Núm. máx. de Puntos-Ruptura
                        Proceso 2—Estab.-Punto-Ruptura
PRO2    PSHS    B,X         Guarda regs. que se alterarán
        LSLA                Multiplica por dos el desplaz.
        LEAX    TABPR,PCR   Dirección de base de la tabla
        LDX     A,X         Toma Dir. en Tabla-Puntos-Rup. (N)
        LDB     ,X          Toma en esa dir. el Opcode
        LSRA                Restaura valor primitivo A
        STB     A,X         Almacena Opcode en Valores-
                            Sustituidos (N)
        LDB     OPSWI,PCR   Toma el Opcode SWI
        STB     ,X          Lo almacena en la dir.
        PULS    B,X,PC      Restaura y retorna
                        Fin del proceso 2
```

Alerta roja

«Flyerfox» es un vertiginoso juego que permite al usuario participar en combates aéreos desde la cabina de un moderno avión de combate

A pesar de que los juegos por ordenador han recorrido un largo camino desde los primeros días de los *Space invaders* (véase p. 1095), la mayor parte de sus innovaciones han estado relacionadas con el desarrollo de gráficos. Los programadores se han preocupado fundamentalmente de hallar nuevas formas de comprimir los datos de un número cada vez mayor de pantallas de gráficos en una cantidad de RAM limitada. Mientras tanto, las excelentes capacidades de sonido de muchos ordenadores personales han sido en gran parte dejadas de lado.

Ahora la empresa norteamericana Tymac ha empezado a distribuir una serie de juegos que incorporan síntesis de voz sin utilizar en realidad ninguna interface especial. El primero de estos programas que se ha comercializado a nivel masivo es *Flyerfox*, para el Commodore 64. Se trata de un programa de simulación de vuelo en el cual el jugador "pilota" un avión de combate que debe escoltar a un Jumbo, a bordo del cual viaja un alto funcionario del gobierno, a través de un espacio aéreo en litigio. Los cazas enemigos intentan derribar el avión comercial, y el objetivo del jugador es entrar en combate con éstos y abatirlos. La síntesis de voz utilizada en el juego consiste en una serie de mensajes que se le transmiten al jugador-piloto desde el Jumbo.

El sintetizador de voz es una parte del software que ocupa alrededor de 11 K de memoria para almacenar los datos que se utilizan para reproducir las frases requeridas. *Flyerfox* emplea el método de codificación de "predicción lineal". En este sistema, las palabras se convierten en señales digitales, que se almacenan luego en RAM. Cuando se necesita una palabra determinada, se accede a los datos digitales correspondientes y la palabra se reproduce a través del chip SID del Commodore.

Los gráficos del juego son siempre en alta resolución. La visualización en pantalla consiste en vistas frontales que muestran el cielo tal como se vería desde la ventana de la cabina, y el panel de instrumentos. El jugador cuenta con varios medios auxiliares de navegación, entre ellos una brújula y un panel de radar que muestra a los cazas MiG aproximándose, al tiempo que se le concede tiempo para prepararse para el combate. Otra de las ayudas que se proporcionan son dos luces intermitentes, una a cada lado del horizonte, que le indican si los MiG se hallan sobre o bajo el nivel de la cabina.

Las escenas de combate son rápidas y muy realistas. Cuando aparece un MiG en la pantalla, el programa produce un zumbido de aviso y el jugador debe entonces maniobrar el *Flyerfox* de modo tal que el atacante quede justo en el punto de mira. Esto no es fácil, ya que los aviones hacen regates y bajan en picado a gran velocidad. Una vez el objetivo está fijado en la mira, el jugador puede disparar los misiles rastreadores de calor; sin embargo, éstos no son infalibles y con frecuencia los cazas enemigos consiguen escapar.

A pesar de que los gráficos son de una gran calidad, no ofrecen excesiva variedad. La ilusión de movimiento se consigue modificando los patrones de las nubes, y el suelo es simplemente una cuadrícula rotatoria. También hay que decir que el avión de línea aporta poco o nada al juego. Dejando de lado el evidente paralelismo con el hecho real del Jumbo 007 surcoreano abatido por un avión soviético hace un tiempo, que deja el juego abierto a ciertas acusaciones de falta de buen gusto, resulta difícil comprender los motivos de la inclusión del avión comercial. Sólo se lo puede observar desde la cola, y el *Flyerfox* es incapaz de darle alcance cuando intenta entrar en combate con los cazas. Además, a diferencia de un avión verdadero, al ser atacado, el Jumbo ni siquiera intenta huir.

Ciertamente, *Flyerfox* representa una nueva tendencia en materia de juegos por ordenador. La síntesis de voz dentro de un programa es un tema que se ha estado considerando durante cierto tiempo. Ahora Tymac ha producido un juego que utiliza la voz, pero que no exige ninguna interface ni dispositivo de hardware especial. En este sentido, es probable que llegue a ser considerado como un hito en el desarrollo de juegos por ordenador.

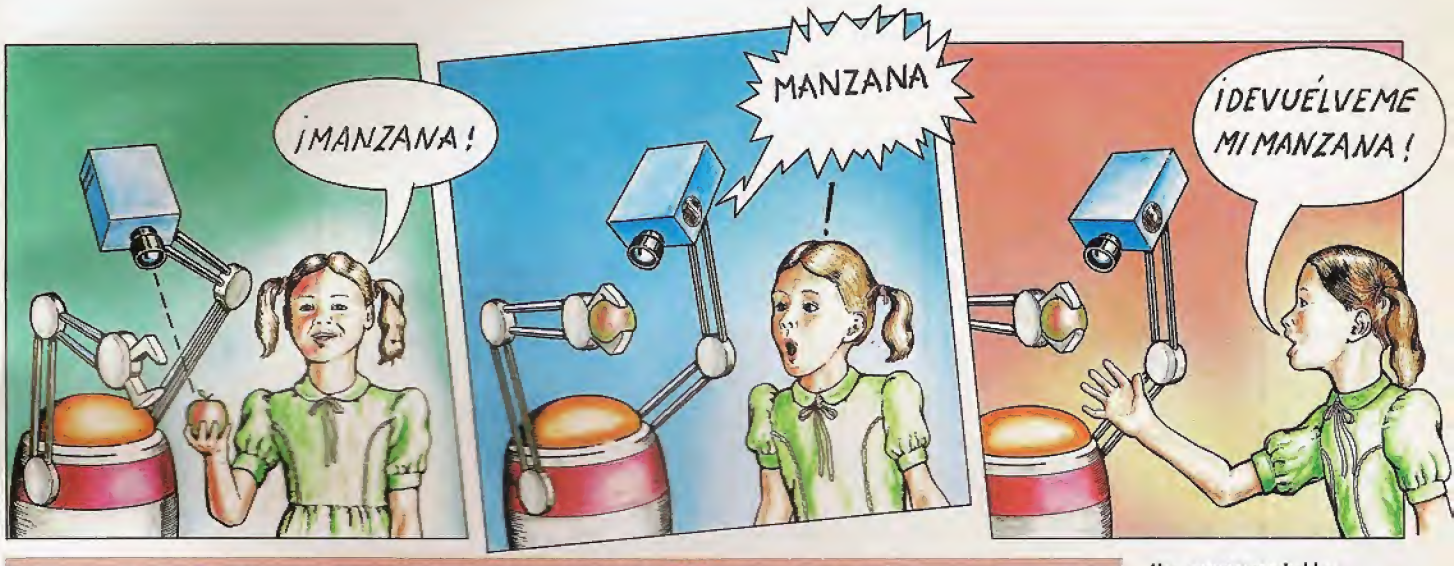
Perseguir y destruir
El piloto del Flyerfox puede reunir información acerca de las posiciones de los cazas enemigos tanto a partir del panel de instrumentos como de la observación del cielo. Los puntos de la pantalla del radar son aviones, si bien no todos ellos atacarán. La altura relativa de los MiG se indica mediante los dos cuadrados blancos que se observan a cada uno de los lados del altímetro. La brújula sirve para volver a localizar el Jumbo

Los MiG atacan

Iain McKinnell

Flyerfox: Para el Commodore 64
Editado por: Tymac Corporation
Autores: Gregory Carbonaro, Charles Teufert, Ronald Pintus, Arthur Aspromatis
Palanca de mando: Necesaria
Formato: Disco o cassette

Hablemos del habla



Steve Cross

Ver, comparar y hablar
Cuando una persona ve un objeto —una manzana, por ejemplo— y le da un nombre, existe una comprensión del significado de "manzana". El robot puede reconocer visualmente el objeto mediante la comparación de lo que ve con una imagen interna, y puede repetir el patrón de sonido que tiene almacenado para acompañar a "manzana". Pero no comprende en absoluto que el objeto es una fruta comestible ni, y quizá sea más importante, que la manzana en realidad le "pertenece" a la persona. Esto, por supuesto, es algo que el humano entiende perfectamente

Para entender la dificultad de hacer hablar a un robot es necesario considerar algunas teorías acerca de la adquisición del lenguaje

El estudio del habla humana ha propiciado la existencia de dos escuelas de pensamiento: la de quienes piensan que las aptitudes del lenguaje son innatas (algo con lo que ya hemos nacido) y la de quienes opinan que el lenguaje se adquiere o se aprende. Los psicólogos que sostienen que éste es innato señalan que el hombre es la única criatura que se comunica mediante el lenguaje. Quienes piensan que es adquirido aluden a experimentos realizados con animales a los cuales se les ha enseñado a comunicarse con los humanos a través de signos.

Si las personas aprendieran a hablar simplemente por hallarse inmersas en un mundo que se relaciona por medio del habla, entonces sería lógico buscar algún método para conseguir que los robots hicieran lo mismo. Al fin y al cabo, la vida sería muchísimo más sencilla si un robot pudiera aprender el lenguaje por el mero hecho de escuchar cómo lo hablan quienes lo rodean.

Se han llevado a cabo ciertos intentos para lograr que un ordenador amplíe sus conocimientos de gramática gracias a que se le proporcionan ejemplos extras de estructuras gramaticales de frases, mientras que con otros experimentos se ha intentado conseguir que un robot aprenda palabras y morfemas (elementos del lenguaje) nuevos en cualquier idioma simplemente mostrándoselos. Pero aún no se ha desarrollado ningún sistema que haya logrado enseñarle a un robot a aprender el habla.

En consecuencia, en la práctica, las aptitudes del robot para adquirir un lenguaje se basan en el supuesto de que éste es innato, que las aptitudes no se aprenden y que lo que debemos hacer es elaborar las reglas del lenguaje y fijárselas al robot permanentemente como si él hubiera nacido con ellas. En general, esto consta de dos fases diferenciadas: el análisis sintáctico y el análisis semántico.

El *análisis sintáctico* trata de la gramática de lo que se está diciendo y decodifica la estructura superficial del mensaje o codifica el mensaje en una forma gramatical correcta para que el robot la transmita. El método más común para realizar esto es mediante un *árbol de análisis* (*parsing tree*), que gradualmente va descomponiendo, o construyendo, una oración a partir de las diversas partes de la misma. Ésta no es una tarea sencilla; no obstante, poco a poco se está abordando con cierto éxito.

El *análisis semántico* es mucho más complicado e implica elaborar el sentido del mensaje (cuando el robot está escuchando hablar a alguien) o elaborar cuál es el mensaje que se necesita comunicar (cuando el robot desea hablarle a uno). El problema del análisis semántico es que el lenguaje no es independiente de su entorno: su significado depende del contexto en el que está incluido (y esto no se aplica sólo al entorno hablado, sino al contexto global del mensaje). Este contexto puede abarcar el conocimiento del mundo circundante mientras uno está hablando, así como el conocimiento que cada una de las partes posee acerca de la otra.

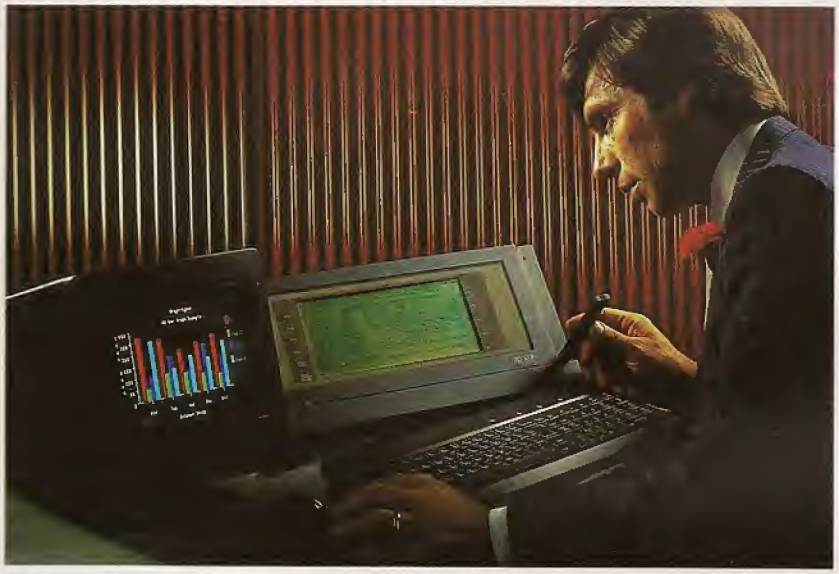
Este enfoque se ha adoptado en experimentos dirigidos por el científico informático Terry Winograd, quien escribió un programa que permitió que un robot entendiera lo que se le decía y actuara a tenor de las instrucciones. Sin embargo, Winograd utilizó una simulación por ordenador de un robot que sólo era capaz de operar en un entorno definido muy minuciosamente. En este caso, el mundo del robot consistía en una cantidad de bloques de construcción que era capaz de manipular. El pro-

grama de Winograd, conocido como SHRDLU, fue capaz de efectuar un buen análisis semántico, pero el entorno que era capaz de discernir era extremadamente simple. Un robot que hubiera estado operando en el caos que es el mundo real hubiese tenido muchísimas más dificultades para entender lo que se le estaba diciendo.

Con el objeto de que los robots empleen provechosamente el lenguaje, ha de haber un mensaje que se transmita de la persona al robot, y viceversa. Para los autómatas cibernéticos, hablar es relativamente fácil, porque, dado que sus conocimientos son tan restringidos, es probable que lo que deseen expresar sea muy limitado. Para ellos es mucho más difícil entender lo que una persona pueda decirles, porque todo lo que se desee comunicarles será mucho más difícil de analizar.

En una época se pensó que la entrada en forma de voz a los robots se analizaría llevando a cabo un análisis sintáctico de la entrada y que éste revelaría el significado del mensaje. Pero trabajos recientes han puesto de relieve la importancia del conocimiento acerca del mundo circundante y del contexto en el cual se expresa el mensaje. Esto ha llevado a experimentos en los cuales se realiza un análisis sintáctico tentativo de la señal del habla para formular una primera conjetura sobre el significado. Luego, a la luz de lo que el robot sabe acerca del mundo y de las cosas que es probable que se digan en su entorno, el robot revisa su análisis sintáctico original con la finalidad de ir acercándose gradualmente a un análisis correcto de lo que se está diciendo. No obstante, esto dista mucho de lo que hoy puede hacer cualquier robot. Ahora vamos a estudiar cómo hablan y cómo entienden la voz los sistemas actuales de robots.

Apricot F1



Oír y hablar
Crear habla por ordenador y habla para robots es bastante fácil. Existen a la venta dispositivos para síntesis de voz, como el Currah, incluso para los ordenadores personales más pequeños. Pero el reconocimiento es más difícil debido a las variaciones de la forma en que los humanos pronuncian los sonidos vocálicos y a causa de la cantidad de memoria y de potencia de proceso que requieren para manejar un vocabulario de más de unas pocas palabras. Sistemas como el Big Ears (literalmente "grandes orejas") y el Apricot F1 llevan incorporada una pequeña gama de instrucciones reconocibles, pero demasiado pocas como para cubrir más de una cantidad mínima de operaciones

Síntesis de voz

El método más simple de síntesis de voz emplea un magnetófono en el cual hay un mensaje pronunciado por un ser humano que está grabado en cinta y que el robot reproduce en un momento adecuado. Puede que esto no se parezca mucho a la idea que se formó usted cuando pensó por primera vez en el habla del robot, pero es el punto de partida de todos los sistemas de síntesis de voz. En primer lugar, analizaremos las limitaciones de este método y luego veremos cómo superarlas.

La limitación más evidente es que un magnetófono es mecánico, caro, voluminoso y susceptible de romperse. De modo que el siguiente paso consiste en tomar el mismo mensaje y convertirlo a una forma digital, de modo que pueda ser almacenado en un chip de la memoria del robot. Ello se realiza mediante un convertidor de analógico a digital, en el cual se utilizan números para representar la forma de onda de la voz, que varía de manera constante. Éste es el mismo procedimiento que se aplica para la grabación digital de música en, por ejemplo, los sistemas de disco compacto.

Este método tiene, asimismo, sus inconvenientes. Uno de los problemas fundamentales es que una señal digitalizada ocupa muchísimo lugar en la memoria. La grabación en disco compacto muestrea la señal acústica alrededor de 44 000 veces por segundo con una resolución de aproximadamente 16 bits (es decir, la amplitud de la forma de onda en cualquier momento se almacena como un número de 16 bits, que permite discernir 2^{16} niveles, donde $2^{16} = 65\,536$). Utilizando este sistema, cada segundo de la grabación ocuparía 88 000 bytes. Evidentemente, un mensaje hablado supera la capacidad de almacenamiento de cualquier microordenador. Sin embargo, esta velocidad de muestreo sólo es aplicable a la reproducción de sonido de alta fidelidad; se podría construir un sistema de habla sencillo con una resolución de ocho bits y una velocidad de muestreo de 3 000 muestras por segundo, ¡lo que sólo ocuparía tres Kbytes!

Cortesía de William Stuart Systems Ltd

Sin embargo, con el fin de dejar libre la máxima cantidad de espacio de memoria, es necesario realizar más economías. Los lingüistas han descubierto que el lenguaje hablado se puede descomponer convenientemente en unidades que se denominan fonemas. En conjunto, la opinión general coincide en el hecho de que existen alrededor de 40 fonemas diferentes para la mayoría de los lenguajes hablados, de modo que es posible almacenar la exacta información acústica necesaria para describir cada

uno de estos 40 fonemas y luego utilizar éstos como la base del habla del robot. Típicamente, la información relativa a los fonemas está retenida en un chip sintetizador de voz que se fabrica comercialmente y todo lo que el robot ha de hacer es encadenar estos fonemas para generar el mensaje requerido. Este mensaje se suele almacenar como una serie de números en la memoria del ordenador.

La mayoría de los sintetizadores de voz que existen en uso se pueden programar escribiendo el mensaje que el robot ha de expresar en una versión fonética del inglés. Por consiguiente, el mensaje "*Can you come here?*" (¿Puedes venir aquí?) se escribiría "kan yew kum heah", y esto sería suficiente para que el chip sintetizador produjera la serie correcta de sonidos. Esta notación no es exactamente la misma que utilizan los lingüistas para describir los fonemas (ellos poseen su propio alfabeto especializado) pero es suficiente para los robots.

Llegados a este punto, observará que el robot ya no está empleando un mensaje pregrabado: está realmente generando mensajes propios. Por esta razón se puede lograr que el robot diga lo que deseemos sin necesidad de tener el mensaje entero grabado con anterioridad.

Por lo tanto, si así nos lo propusiéramos, podríamos tratar de programar algunas de las reglas de la gramática en un intento de hacer que el robot dijera cosas originales. Pero, como ya hemos mencionado, el número de cosas diferentes que podría expresar un robot es más bien limitado, de manera que no existe ninguna necesidad de una complejidad excesiva, a menos que nos dejemos llevar por un espíritu aventurero o por la curiosidad de ver qué es lo que se puede hacer.

Si ha escuchado alguna vez un sintetizador de voz en un robot, sabrá que la calidad de la voz, si bien por lo general es comprensible, de ningún modo es perfecta. Ello se debe a dos factores. El primero es que la forma que toma un fonema cuando lo utiliza un humano varía considerablemente en virtud de los fonemas que lo preceden y le siguen. El segundo factor es que el sonido de la voz humana varía según el significado que deseemos transmitir. "¿Quieres sentarte?" y "¿Quieres *sentarte*?" son dos mensajes escritos de manera idéntica, pero sonarán muy distintos si el primero lo dice un cortés anfitrión a su huésped y el segundo lo pronuncia un maestro enojado. Se han realizado algunos intentos por captar esta entonación en los sistemas de síntesis de voz, pero su aplicación es difícil, dado que un robot no conoce el significado de las palabras que está pronunciando.

Reconocimiento de voz

El problema inevitable a resolver cuando se desarrolla un sistema de reconocimiento de voz es que las cosas que desearíamos decirle a un robot, y las formas de expresarlas, son muchas y muy diversas. Un enfoque posible a este problema sería utilizar una grabación en cinta de todo lo que quisiéramos que el robot comprendiera. Cuando habláramos, él podría entonces explorar todas sus grabaciones en cinta y buscar la que más se pareciera al mensaje que acabara de escuchar; y así es, en principio, como reconocen la voz muchos robots. Almacenan "modelos" internos de mensajes hablados y, cuando se les habla, simplemente buscan el modelo que ofrece la mayor similitud. Estos modelos generalmente se obtienen entrenando al robot (repitiendo varias veces una palabra o frase) hasta que adquiera un modelo "promedio" de lo que hemos dicho. Este método funciona bien si sólo se tiene una pequeña cantidad de cosas que decirle al robot y si vamos a decírselas siempre más o menos de la misma forma. Se utiliza para aquellos robots que responden a instrucciones sencillas, tales como "adelante", "gira a la izquierda", etc.

Sin embargo, éste es un problema comparativamente simple que se conoce como *reconocimiento de voz discreta*, porque cada ítem hablado es "discreto", es decir, está separado de otros mensajes mediante una breve pausa durante la cual no se dice nada.

El verdadero problema surge cuando deseamos hablarle al robot empleando *voz continua*, que es la que utilizamos normalmente. Pruebe decir "Es un hermoso día de verano", y escuche atentamente lo que ha pronunciado. Descubrirá que se oye algo así como "Esunher mosodía deverano", con las palabras y los sonidos chocando unos contra otros.

La forma en la cual una persona aborda este problema cuando está escuchando hablar a otra consiste en adivinar qué es lo que el interlocutor intenta decir (lo que no suele ser difícil) y utilizar esta conjetura para decodificar el mensaje. Pero para que un robot hiciera eso, habría que saber muchísimo acerca de qué es probable que se diga y quiera decirse con ello: una tarea muy complicada.

En general, el empleo de la síntesis de voz en robots se está convirtiendo en algo común, si bien aún quedan cosas por hacer en cuanto a elevar la calidad de su habla. El reconocimiento de voz es una tarea mucho más difícil y, en la actualidad, lo más que se puede lograr con cierta facilidad es dotar al robot de una comprensión del habla equivalente a la de un perro entrenado que responda a las instrucciones habladas, siempre y cuando éstas no sean muy numerosas. No obstante, existe un extraordinario interés por resolver todos los problemas que entraña el habla del robot.

Óyeme, siénteme
El Voicemate es un brazo-robot controlado por voz que desarrolló el departamento de ingeniería del Politécnico de Newcastle (Gran Bretaña) para uso industrial y de laboratorio

Cortesía de Newcastle Polytechnic

Campo de fútbol

En esta ocasión examinaremos el paquete "Multiplan", una completa hoja de trabajo electrónica para el Commodore 64

En *Multiplan*, primer programa de hoja electrónica producido por Microsoft, se han incorporado muchas ideas avanzadas, desarrolladas a partir de paquetes de hoja electrónica anteriores. Éstas incluyen la capacidad de actuar sobre grupos de celdas que se describen por nombre; clasificar un grupo de entradas de acuerdo a un criterio especificado; dividir pantallas, permitiendo visualizar simultáneamente zonas diferentes de la hoja electrónica; buscar rápidamente en la información retenida en forma de tabla y luego producir un valor requerido, y llevar a cabo construcciones condicionadas IF...THEN, entre otras. Disponible originalmente sólo para ordenadores de alto nivel tales como el IBM PC y el Apple II, recientemente ha salido al mercado el *Multiplan* para el Commodore 64.

El modelo que construiremos con este programa simplifica la tarea de mantener al día las estadísticas. Los datos que utilizamos se refieren al fútbol americano, pero la estructura se puede adaptar para otros deportes.

Después de cargar el *Multiplan*, se visualiza una hoja de trabajo de formato estándar de 63 columnas y 255 filas. Tanto las filas como las columnas están numeradas, de modo que la celda base, en la esquina superior izquierda, es la celda R1C1 (por Fila (Row) 1 Columna 1). En la parte inferior de la pantalla aparece un menú de instrucciones, con un cursor que destaca la primera opción, Alpha. Las instrucciones del menú de *Multiplan* se pueden seleccionar pulsando la barra espaciadora para desplazar el cursor hasta la instrucción deseada y pulsando Return, o tecleando la primera letra de ésta.

Con frecuencia, la selección de una instrucción hace que se visualice un submenú que ofrece una amplia variedad de opciones para formateo de datos, administración de memoria, etc. Pulsando una tecla de letra se accede a una instrucción, de modo que uno debe digitar A de Alpha antes de entrar texto. Los números se pueden entrar directamente, pero las fórmulas deben ir precedidas por un signo más (+) o igual (=).

Las dos primeras filas de la hoja de trabajo retienen títulos. Por conveniencia, hemos formateado las celdas desde R1C1 hasta R2C5 para texto continuo, lo que permite extender el texto más allá de los límites de la celda. Ello se consigue digitando:

F(formato) C(eldas) R1C1:R2C5

colocando luego el cursor encima de la palabra Cont y pulsando Return. Los dos puntos se utilizan para indicar una serie de celdas. Algunas columnas han variado su ancho para dar cabida a sus entradas.

La hoja de trabajo tiene dos porciones principales: una retiene información para un equipo específico durante un período de nueve semanas, y la otra es una tabla de los resultados gana/pierde para todos los equipos de la misma "conferencia" (véase texto sobre fútbol americano en el margen de la página contigua). Una vez construido el esqueleto del modelo, gran parte de los datos semanales habrán de entrarse a mano, con unas pocas fórmulas para mantener actualizados los totales a medida que avanza la temporada.

Resultados de la Liga

Informe del rendimiento por equipos

Ian McKinnell

La primera porción de la hoja de trabajo es una tabla. Los totales de ésta se deben actualizar cada semana, después de que hayan jugado los equipos. Mediante el almacenamiento de la información en una tabla podemos sacar partido de una de las características de *Multiplan*: la facilidad SORT (clasificar). Hemos entrado los nombres, categorías y datos de los equipos. El orden inicial de éstos está basado en las posiciones que ocupan en la liga. Sin embargo, la tabla se puede clasificar por cualquiera de las categorías almacenadas. *Multiplan* clasificará una serie especificada de filas de una columna por orden numérico ascendente o descendente. El texto se clasifica alfabéticamente.

Como ejemplo de función SORT, vamos a reacomodar los datos reflejados por los nombres de los equipos en orden alfabético. Después de digitar S, de SORT, *Multiplan* visualiza lo siguiente:

SORT by column:__between rows:__and:__
order: >< (CLASIFICAR la columna__entre las filas__ y__por orden ><)

Nosotros queremos clasificar la columna 1 entre las filas 7 y 21 en orden ascendente (>). La pulsación de Return hace que *Multiplan* reorganice los nombres alfabéticamente y reacomode los datos en consecuencia. Por ejemplo, todos los números relativos a Miami se desplazarán junto con Miami hasta su nuevo posición. Simplemente cambiando la columna clave de SORT podemos reacomodar la tabla en función del equipo con mayor puntuación, los equipos que han permitido que sus oponentes marquen la menor cantidad de puntos, etc.

Tabla clasificada

Ahora desplazamos los datos de la pantalla pulsando la flecha del cursor que señala hacia abajo y hallamos la segunda porción de la hoja de trabajo: el informe del rendimiento individual por equipos. Aquí se utilizarán dos fórmulas. La primera es una simple fórmula SUM (suma), para llevar un total actualizado de los valores semanales. Localice la columna etiquetada TOTALS (totales) de la sección dos (R24C12). Desearemos que *Multiplan* sume los valores de cada columna semanal. Dado que queremos copiar la fórmula de modo que se hallen los totales para todas nuestras categorías (cubriendo la zona desde R25C12 hasta R32C12), necesitamos incorporar una referencia a una celda relativa. En *Multiplan* esto se realiza señalando las celdas activas con el cursor. La fórmula se entra digitando: =SUM(y pulsando luego la tecla de la flecha hacia la izquierda hasta que el cursor quede situado en R25C2. Entonces digitamos el carácter dos puntos para indicar que se está especificando una serie de celdas. El cursor regresa automáticamente hasta la celda en la cual uno está entrando la fórmula, de modo que pulse una sola vez la flecha hacia la izquierda, con el cursor situado en R25C11, y después pulse Return. La fórmula se verá así:

=SUM(R[-6]C:R[-1]C)

y se visualizarán los totales para los valores retenidos en la gama descrita. Ahora copie la fórmula en la serie de celdas desde R26C12 hasta R32C12 manteniendo el cursor en la fórmula y empleando la instrucción COPY: C(opiar) d(own) (abajo) 7 filas. Utilice el mismo proceso para hallar los totales para YDS RUSH e YDS PASS. La fórmula SUM se coloca en la celda R27C3, para las yardas ganadas en ataque, y en R31C3, para las yardas cedidas al equipo rival. La fórmula se copia en las ocho columnas de la derecha, para cubrir el período completo de nueve semanas.

La segunda fórmula, empleando la sentencia IF, es un poco más complicada pero sumamente útil. En nuestro modelo dejaremos que Multiplan determine si un partido se ha ganado o se ha perdido mediante la comparación de los puntos totales de dos categorías: Points Scored ("puntos ganados" por nuestro equipo) y Points Allowed ("puntos cedidos": el marcador del rival). Necesitamos una sentencia como ésta: If Puntos Ganados > Puntos Cedidos, print WIN (ganado), else print LOSS (perdido).

Nuevamente, queremos referencias relativas y, por lo tanto, utilizamos movimientos del cursor para indicar las posiciones de los dos valores. Coloque el cursor en R34C3, etiquetada WIN/LOSS, y entre la fórmula:

IF(R[-6]C>R[-2]C,"WIN","LOSS")

Observe que el texto que se utiliza dentro de las fórmulas debe ir encerrado entre dobles comillas y que se necesitan paréntesis para encerrar las condiciones. Ahora copie la fórmula a lo largo de la fila,

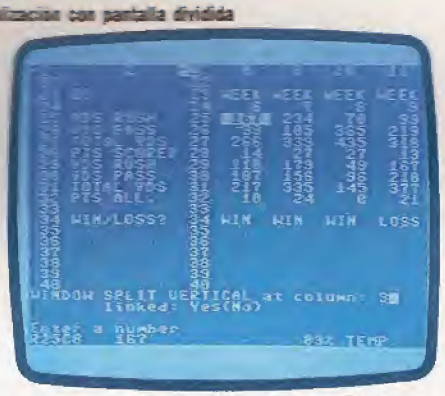
Construcción condicional (IF...THEN)

como antes. El modelo que hemos creado contiene datos para nueve semanas. Debido a las dimensiones de la pantalla, no podemos ver ni las etiquetas del margen izquierdo de nuestro informe por equipos, ni los totales de la derecha. Pero podemos dividir la pantalla en dos ventanas, que se puedan desplazar juntas o por separado.

Ahora desearemos dividir la pantalla verticalmente en la columna 3, de modo que pulse W(indow) (ventana) y S(plit) (dividir), seguidas de V(ertical). Entonces el programa visualizará:

WINDOW SPLIT VERTICAL at column:__linked YES NO (División vertical de ventana por la columna__unidas SI NO)

Visualización con pantalla dividida



Usted tendrá que entrar la columna 3, desplazar el cursor hasta NO y pulsar Return. Si las ventanas no están unidas se les puede desplazar por separado. Ahora las etiquetas se pueden ver independientemente de qué porción de la hoja de trabajo se esté visualizando. Para cerrar la ventana digite W(indow) (ventana) C(lose) (cerrar), seguidas de su número.

Kevin Jones

Fútbol americano

Para quienes no estén familiarizados con el fútbol americano, he aquí una breve explicación de los términos que se utilizan en él. Dos equipos compuestos por 11 jugadores se turnan para tratar de desplazar un balón ovoide a través de una línea de gol. Las metas opuestas están separadas por 100 yardas (91 m). El balón lo puede transportar un corredor, arrojándolo hacia adelante como un pase, o pateándolo por entre los postes de la portería. Se consiguen tres puntos por un *kick*: patada (llamado *field goal*: tiro libre); se otorgan seis puntos por un *carry* (denominado *touchdown*: tanto marcado al tocar el suelo con el balón detrás de la meta del adversario), y uno por un *kick* después de un *touchdown* (lo que se conoce como un punto tras *touchdown*).

Cada equipo dispone de cuatro intentos o *downs* para llevar el balón 10 yardas hacia la meta contraria. Si tiene éxito, puede seguir hacia la meta con otros cuatro *downs*. Cuando un jugador lleva el balón, ello se denomina un *rush* (carrera), y el número de yardas que recorra un equipo constituye un índice de cuán lejos se ha llevado el balón durante el partido. El número de yardas *passing* señala la distancia a la cual se ha lanzado el balón.

En la National Football League (la principal liga profesional de Estados Unidos) hay dos "conferencias": la American Conference y la National Conference. Los equipos juegan una temporada de 16 encuentros que comienza en septiembre y termina en enero con la *Super Bowl* (Supertazón: por la forma del típico estadio de fútbol). En ésta participa el mejor equipo de cada conferencia

Aventura y emoción

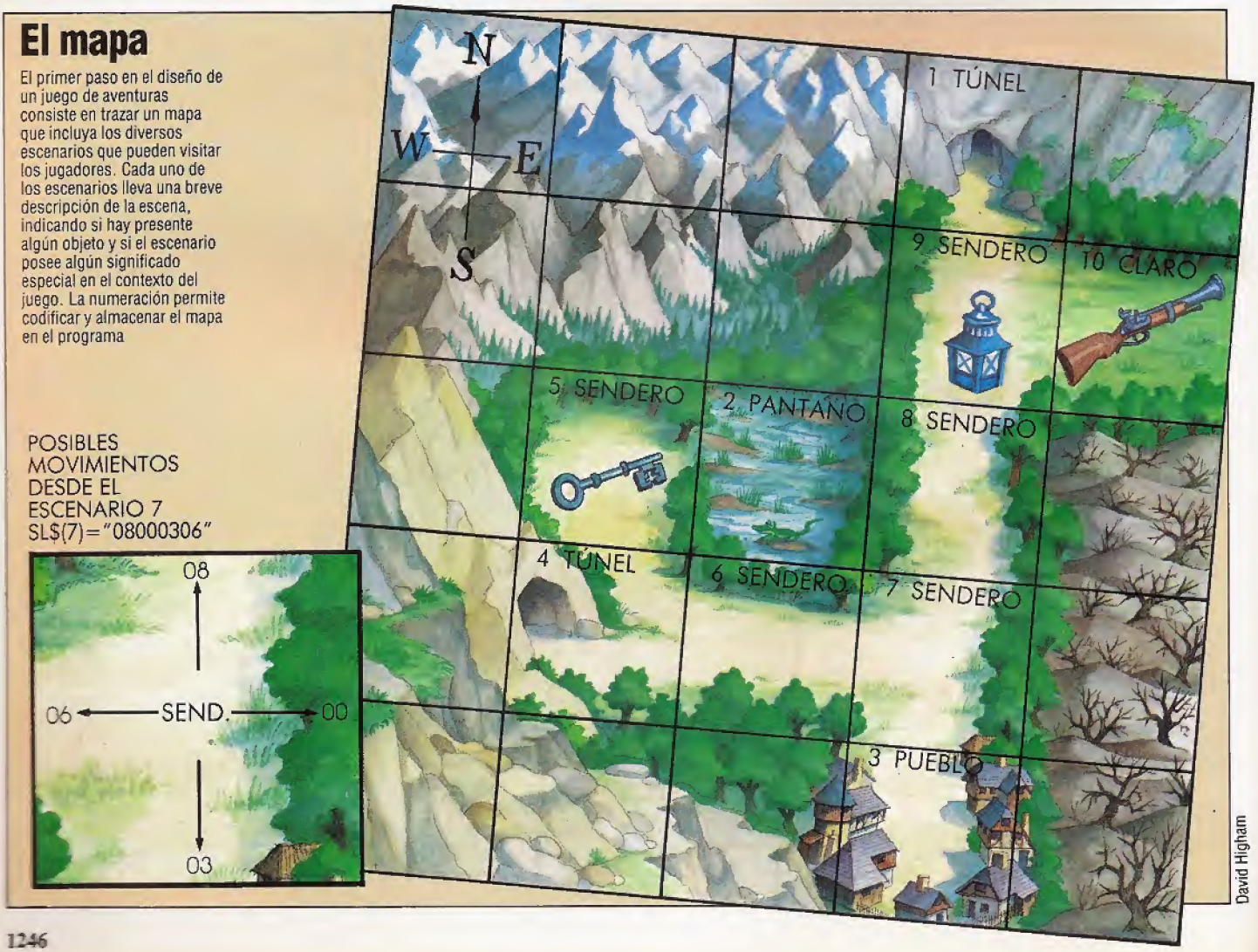
Iniciamos un gran proyecto en que guiaremos al lector a través de las etapas de creación de un juego de aventuras

Los juegos de aventuras se hicieron populares a principios de los años setenta, cuando se ideó el juego *Calabozos y dragones* (*Dungeons and dragons*). En este juego los participantes asumen el papel de varios personajes de un mundo imaginario diseñado por el "amo de la mazmorra" (*dungeon master*). Este mundo imaginado se suele componer de un intrincado laberinto de salas que contienen objetos y peligros que han de ser superados por los jugadores. Generalmente, el objetivo del juego no es otro que escapar del laberinto, en la mayoría de los casos rescatando por el camino a alguien o algo. Los programadores de ordenadores centrales fueron los primeros que aplicaron el juego a éstos, construyendo complejos laberintos para que deambularan por ellos otros usuarios de ordenadores centrales. La ventaja del *Calabozos y dragones* basado en ordenador era que el amo de la mazmorra y los jugadores no tenían necesariamente que estar presentes al mismo tiempo, permitiendo que las personas jugaran en cualquier momento que lo desearan. Desde entonces, el juego del tipo *Calabozos y dragones* ha ensanchado considerablemente tanto su horizonte como su atractivo: el MUD (*Multi-User Dungeon*: mazmorra multiusuario; véase p. 864) es un buen ejemplo del grado de refinamiento que han alcanzado algunos de ellos.

Algunos juegos de aventuras están basados exclusivamente en texto, mientras que otros se valen del color y los gráficos para proporcionar imágenes en pantalla. Sin embargo, algunos críticos sostienen que la adición de gráficos supone la ocupación de un valioso espacio de memoria que, de lo contrario, se podría utilizar para agregar más enredos a la estructura del juego. Asimismo, señalan que la imagen gráfica por ordenador de una escena o de un lugar no tiene comparación posible con la propia imaginación de uno al evocar una imagen basada

El mapa

El primer paso en el diseño de un juego de aventuras consiste en trazar un mapa que incluya los diversos escenarios que pueden visitar los jugadores. Cada uno de los escenarios lleva una breve descripción de la escena, indicando si hay presente algún objeto y si el escenario posee algún significado especial en el contexto del juego. La numeración permite codificar y almacenar el mapa en el programa



POSIBLES MOVIMIENTOS DESDE EL ESCENARIO 7 SL$(7)="08000306"

David Higham

en una descripción textual. No obstante, este aumento de popularidad de los juegos de aventuras se puede atribuir casi con total seguridad al mayor atractivo visual que otorgan los gráficos y, si bien algunos recientes juegos para micros sólo utilizan imágenes sencillas para mejorar el texto, otros intentan dar mayor relevancia a la vertiente visual.

En nuestro proyecto de programación analizaremos las técnicas que se emplean en la programación de un juego de aventuras. A lo largo del proyecto se le irán proporcionando secciones de un listado para un juego de aventuras llamado *Digitaya*, que irán construyendo un programa completo. En este juego, el participante asume el papel de un agente "electrónico" al que se le encarga la tarea de descender al interior de un microordenador para localizar y rescatar al misterioso Digitaya de las garras de la máquina. A lo largo del camino se presentan muchas dificultades y acechan muchos peligros, y el jugador habrá de valerse de todos sus conocimientos sobre los ordenadores para escapar ileso. El programa, en la medida de lo posible, está escrito en BASIC "estándar", ofreciéndose los *complementos* en los casos necesarios. Por lo tanto, siempre y cuando se disponga de suficiente capacidad de memoria, el programa funcionará en su ordenador. Puesto que vamos a estudiar con todo detalle las diversas técnicas de programación, sería difícil no revelar muchos de los secretos del juego, y ello iría en detrimento, en cierta medida, del placer de jugar con él cuando estuviera acabado. Por consiguiente, construiremos, juntamente con *Digitaya*, un juego paralelo, más breve, llamado *El bosque encantado*, con el que demostraremos las técnicas y los algoritmos utilizados para construir el juego más extenso.

El punto de partida para el diseño de nuestro juego de aventuras es el trazado de un mapa del mundo fantástico que estamos imaginando. En este mapa hemos de marcar los diversos escenarios del mundo, la posición de todos los objetos que se puedan hallar en los mismos, y dar importancia a aquellos escenarios que se consideren "especiales". La mayoría de los escenarios del mapa simplemente permitirán que el jugador entre y salga de ellos y mueva o abandone cualquiera de los objetos que haya en ellos. Los escenarios especiales pueden ser peligrosos (un pantano o un lugar donde habita un dragón) o pueden exigir la realización de una serie de acciones especiales antes de que uno pueda penetrar en ellos o salir de los mismos.

La mejor forma de empezar a hacer un mapa consiste en considerar en líneas generales cuántos escenarios se necesitan para el juego. *El bosque encantado* tiene 10 escenarios y se diseñó en una cuadrícula de 5 × 5 (como refleja la ilustración), mientras que *Digitaya* posee alrededor de 60 escenarios y se diseñó en una cuadrícula de 10 × 10.

Los cuadrados de la cuadrícula inicialmente no tienen números y el diseñador comienza por rellenar el mapa con escenarios. En el mapa de *El bosque encantado* hay un sendero, dos túneles, un pantano, un claro y un pueblo. También están marcados, en la parte inferior de los cuadrados en los que están ubicados, las posiciones de varios objetos. Aquellos escenarios señalados con un asterisco (*) son "especiales" y se los tratará de forma distinta que al resto de los mismos.

Una vez terminado el trazado podemos numerar cada uno de los escenarios. La única consideración especial que hemos tenido en cuenta al elegir el número de escenarios es que todos los especiales se han numerado primero. El orden por el cual se numeren los otros carece de importancia, pero una vez que se han seleccionado los números es importante no modificarlos con posterioridad.

Programación de los datos

La primera tarea de programación es convertir la información del mapa en datos para el programa. Existen muchas formas de hacer esto, pero lo que haremos aquí es utilizar dos matrices unidimensionales para retener los datos. La primera matriz, ES$(), retiene descripciones de cada *es*cenario. Por ejemplo, para el escenario 7, ES$(7) contendrá "en un sendero". Cuando posteriormente los datos se utilicen en el programa para describir un escenario irán precedidos por las palabras "Ud. se halla".

La segunda matriz, SL$(), retiene datos relativos a los posibles movimientos que se puedan efectuar desde un escenario (*sal*idas). Nuestros dos juegos se limitan a cuatro direcciones: Norte, Este, Sur y Oeste. SL$() proporciona información acerca del número de escenario al cual uno se traslada para cada una de las cuatro direcciones. Los datos están almacenados como una serie compuesta por ocho dígitos. El número de escenario para cada dirección se entra por el orden NESO, utilizando un número de dos dígitos para cada dirección.

Por ejemplo, el escenario 7 tiene salidas al Norte, al Sur y al Oeste, pero ninguna hacia el Este. Los dos primeros dígitos de SL$(7) son 08 (no 8), que indica que hacia el Norte está el escenario 8. El segundo par de dígitos, 00, indica que en esta dirección no hay ninguna salida (Este). Los pares de dígitos 03 y 06 representan los escenarios que se encuentran hacia el Sur y hacia el Oeste del emplazamiento 7. Utilizando este sistema se podrían especificar hasta 39 escenarios; si se necesitaran más de 39, entonces los datos para SL$() se habrían de entrar como grupos de tres dígitos.

Los tres objetos de *El bosque encantado* se leen en otra matriz, IV$(,). Esta matriz bidimensional lleva el registro de la posición de cada objeto mientras el mismo se va trasladando a través del bosque. Cada objeto posee una descripción y su emplazamiento inicial en el mapa. Por ejemplo, IV$(C,1) es ESCOPETA, y su posición al comienzo del juego viene dada por IV$(C,2). A medida que, en el transcurso del juego, los objetos se vayan trasladando de un lugar a otro, los elementos de posición de la matriz se irán actualizando.

Al final de los datos del mapa, en nuestros dos listados hay otros datos. Se trata de una "suma de control" y se proporciona para asegurar que los datos de dirección se hayan digitado correctamente. Ello se realiza calculando un total actualizado de los valores de los datos, que se cotejan con la suma de control. Si éstos no coinciden, entonces se ha producido un error y la ejecución del programa se interrumpirá. Usted observará que en *Digitaya* se utilizan dos sumas de control. Ello se debe a que la suma total de los datos de dirección es demasiado grande como para retenerla fácilmente en una sola suma de control, de modo que se calcula por separado un total para los cuatro dígitos izquierdos y los cuatro derechos. En el próximo capítulo del proyecto diseñaremos rutinas para manipular y visualizar los datos de este mapa.

Digitaya

```
6090 REM **** S/R LEER DATOS MATRIZ ****
6100 REM ** LEER INVENTARIO **
6110 DIM IV$(3,2)IC$(4)
6120 FOR C=1TO8
6130 READ IV$(C,1),IV$(C,2)
6140 NEXT C
6150 :
6160 REM ** LEER DATOS ESCENARIOS & SALIDAS **
6170 DIM ES$(55),SL$(55)
6180 C1=0:C2=0: REM INICIALIZAR SUMAS DE CONTROL
6190 FOR C=1T054
6200 READ ES$(C),SL$(C)
6210 C1=C1+VAL(LEFT$(C),4))
6220 C2=C2+VAL(RIGHT$(SL$(C),4))
6230 NEXT C
6240 READ CA:IFCA<>C1 THEN PRINT"ERROR SUMA DE CONTROL":
     STOP
6250 READ CA:IFCB<>C2 THEN PRINT"ERROR SUMA DE CONTROL":
     STOP
6260 RETURN
6270 REM **** DATOS DE INVENTARIO ****
6280 DATA NUMERO DE DIRECCION,45,LLAVE,34,ESCUDO
     LASER,25
6290 DATA BILLETE AL TRIESTADO,26,TARJETA DE CREDITO DE
     DATOS,28
6300 DATA .DIGITAYA,30,LIBRO DE CODIGOS,19,DISPOSITIVO
     ACTIVADOR DEL BUFFER,13
6310 :
6320 REM **** DATOS ESCENARIOS & SALIDAS ****
6330 DATA EN LA TOMA DEL TELEVISOR,00000000
6340 DATA EN LA PUERTA PARA EL USUARIO,00090100
6350 DATA EN LA PUERTA PARA CASSETTE,00110000
6360 DATA EN LA PUERTA PARA PALANCA DE MANDOS,
     00130000
6370 DATA EN UN DISPOSITIVO TRIESTADO,00170000
6380 DATA EN LA UNIDAD ARITMETICA Y LOGICA,00310016
6390 DATA EN EL ACCESO A LA MEMORIA,00490000
6400 DATA EN LA VIA DE E/S,09000001
6410 DATA EN LA VIA DE E/S,10000802
6420 DATA EN LA VIA DE E/S,11000900
6430 DATA EN LA VIA DE E/S,12001003
6440 DATA EN LA VIA DE E/S,13531100
6450 DATA EN LA VIA DE E/S,14001204
6460 DATA EN LA VIA DE E/S,15001300
6470 DATA EN LA VIA DE E/S UNA SEÑAL DICE "S OUT H",
     00001400
6480 DATA EN EL REGISTRO DE DATOS,00061700
6490 DATA EN UNA VIA DE 8 PISTAS,16001805
6500 DATA EN UNA VIA DE 8 PISTAS,17001900
6510 DATA EN UNA VIA DE 8 PISTAS,18002000
6520 DATA EN UNA VIA DE 8 PISTAS,19292100
6530 DATA EN UNA VIA DE 8 PISTAS,20282200
6540 DATA EN UNA VIA DE 8 PISTAS,21272300
6550 DATA EN UNA VIA DE 8 PISTAS,22262400
6560 DATA EN UNA VIA DE 8 PISTAS,23250000
6570 DATA EN LA MATRIZ DE CARACTERES,26360024
6580 DATA EN LO ALTO DE LA MEMORIA,27352523
6590 DATA EN LA MITAD DE LA MEMORIA,28342622
6600 DATA EN LA MITAD DE LA MEMORIA,29332721
6610 DATA ABAJO EN LA MEMORIA,00542820
6620 DATA EN LA GUARIDA DEL ACUMULADOR,00000600
6630 DATA EN UN LARGO CORREDOR,00420006
6640 DATA EN UN REGISTRO INDICE,31000000
6650 DATA ABAJO EN LA MEMORIA,54403428
6660 DATA EN LA MITAD DE LA MEMORIA,33393527
6670 DATA MUY EN LO ALTO DE LA MEMORIA,34383626
6680 DATA EN LA MATRIZ DE CARACTERES,35370025
6690 DATA EN UNA TABLA VECTORIAL ALEATORIA,00000000
6700 DATA EN LO ALTO DE LA MEMORIA CON VISTA A UNA VIA,39003735
6710 DATA EN LA MITAD DE LA MEMORIA,40003834
6720 DATA EN LA MEMORIA — HACIA EL ESTE HAY UN ACCESO,
     41003933
6730 DATA ABAJO EN LA MEMORIA,00004054
6740 DATA EN UN CORREDOR,00430031
6750 DATA EN UN CORREDOR,00440042
6760 DATA EN UN CORREDOR,00004543
6770 DATA EN EL REGISTRO DE DIRECCIONES,00004600
6780 DATA EN UNA VIA DE 16 PISTAS,45004700
6790 DATA EN UNA VIA DE 16 PISTAS,46004800
6800 DATA EN UNA VIA DE 16 PISTAS,47004900
6810 DATA EN UNA VIA DE 16 PISTAS SE VISLUMBRA HACIA EL OESTE UN ENORME
     PORTALON,48005007
6820 DATA EN UNA VIA DE 16 PISTAS,49005100
6830 DATA EN UNA VIA DE 16 PISTAS,50005200
6840 DATA EN UNA VIA DE 16 PISTAS,51000000
6850 DATA EN UN VECTOR A LA MEMORIA,00290012
6860 DATA ABAJO EN LA MEMORIA,00413329
6870 REM ** DATOS SUMAS DE CONTROL **
6880 DATA 100169,103973
```

El bosque encantado

```
6000 REM **** LEER DATOS OBJETOS Y MAPA ****
6010 DIM IV$(3,2),ES$(10),SL$(10),IC$(2)
6020 FOR C=1 TO 3
6030 READ IV$(C,1),IV$(C,2)
6040 NEXT C
6050 :
6060 FOR C=1 TO 10
6065 READ ES$(C),SL$(C)
6070 CC=CC+VAL(SL$(C)):REM TOTAL SUMA CONTROL
6080 NEXT C
6090 :
6100 READ CD:IFCD<>CC THENPRINT"ERROR EN SUMA DE CONTROL":STOP
6110 :
6120 REM ** DATOS OBJETOS **
6130 DATA ESCOPETA,10,FAROL,9,LLAVE,5
6140 :
6150 REM ** DATOS MAPA **
6160 DATA JUNTO A LA ENTRADA DE UN TUNEL,00000900
6170 DATA EN UN PANTANO,00000000
6180 DATA EN UN PUEBLO,07000000
6190 DATA JUNTO A LA ENTRADA DE UN TUNEL,05060000
6200 DATA EN UN SENDERO,00020400
6210 DATA EN UN SENDERO,02070004
6220 DATA EN UN SENDERO,08000306
6230 DATA EN UN SENDERO,09000702
6240 DATA EN UN SENDERO,01100800
6250 DATA EN UN CLARO,00000009
6260 REM ** DATOS SUMA DE CONTROL **
6270 DATA 32253121
6280 RETURN
```

Complementos al BASIC

Nuestros listados se escribieron para el Commodore 64. Para el Spectrum y el BBC Micro es necesario introducir estas modificaciones:

Spectrum:
Para el listado de *Digitaya*, realice estos cambios:

```
6110 DIM V$(8,2,24):DIM I$(4,24)
6170 DIM L$(55,52):DIM E$(55,8)
6200 READ L$(C),E$(C)
6210 LET C1=C1+VAL(E$(C)(TO 4))
6220 LET C2=C2+VAL(E$(C)(LEN(E$(C))−3 TO))
```

Además, todos los valores DATA se deben encerrar entre comillas, con la excepción de la suma de control de la línea 6880. En el listado de *El bosque encantado* introduzca estas modificaciones:

```
6010 DIM V$(3,2,5):DIM L$(10,22)
6015 DIM E$(10,8):DIM I$(2,5)
6030 READ V$(C,1)V$(C,2)
6065 READ L$(C),E$(C)
6067 LET CC=0
6070 LET CC=CC+VAL(E$(C))
```

Encierre entre comillas los valores DATA, a excepción de la suma de control de la línea 6270.

BBC Micro:
En el listado de *El bosque...* agregue esta línea:

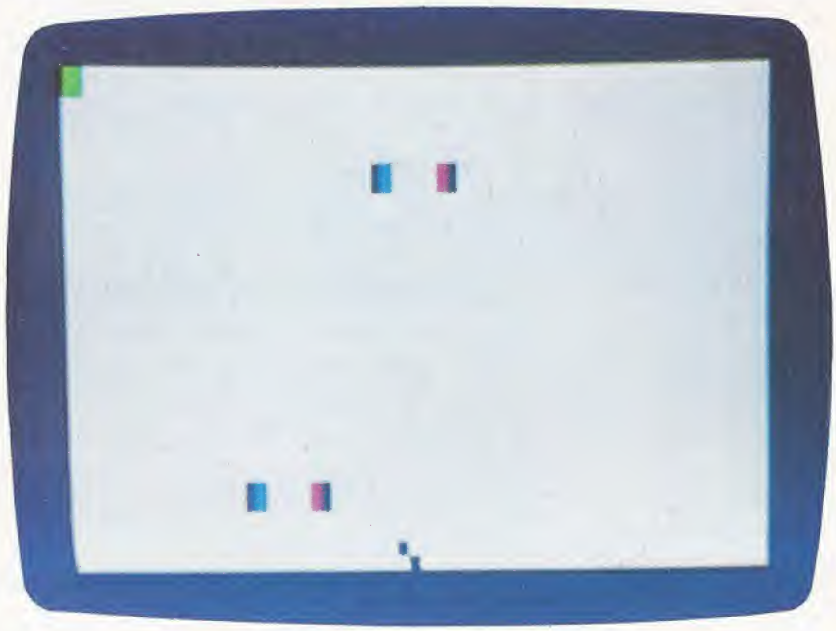
```
6067 CC=0
```

En el de *Digitaya* no se precisan modificaciones.

Slalom

Los juegos deportivos ejercen gran atracción en la mayoría de los usuarios. Presentamos un «juego de invierno» en una versión para el microordenador Dragon

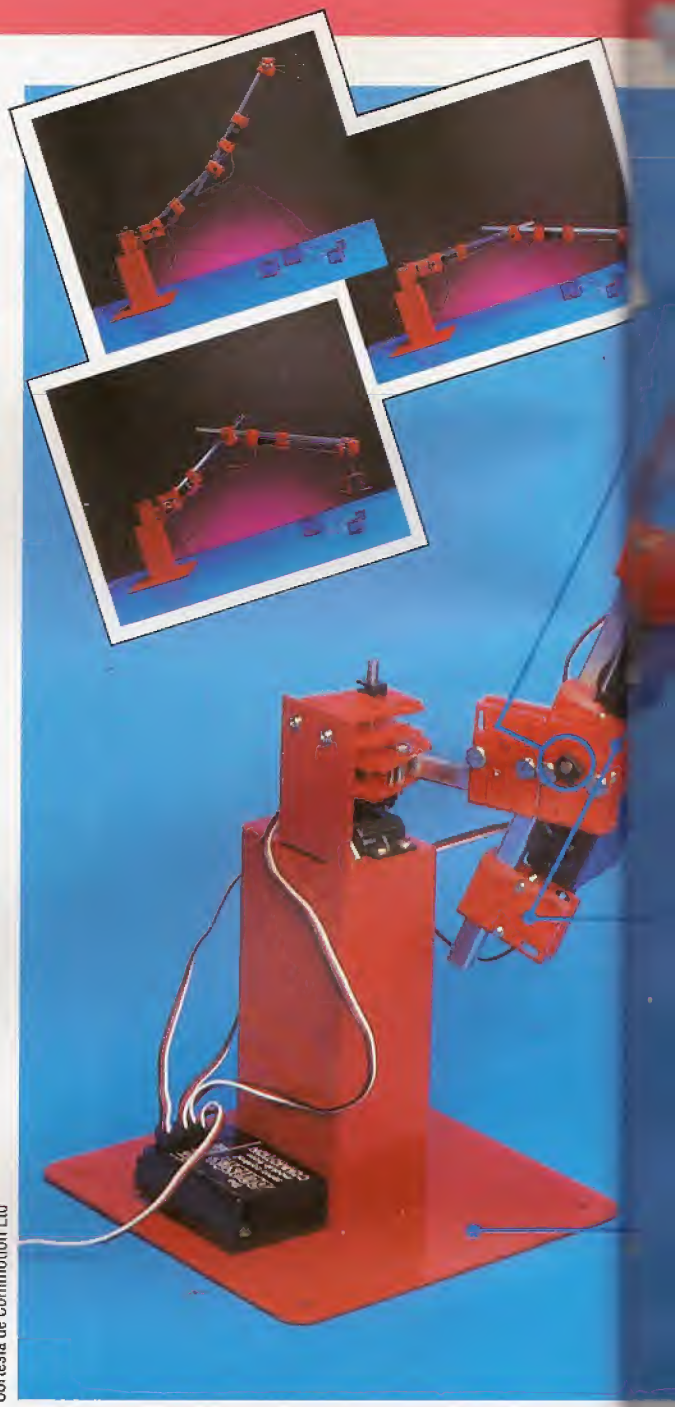
¡Dispóngase a practicar los deportes de invierno sin correr el riesgo de romperse una pierna! Láncese desde lo alto de la pista e intente pasar por el mayor número posible de puertas sin chocar con los palos. Para cambiar de dirección pulse cualquier tecla.



```
10 REM ----------
20 REM "SLALOM"
30 REM ----------
34 REM
35 REM INICIALIZACION
36 REM
38 REM TABLA DE POSICIONES
39 REM DEL ESQUIADOR
40 DIM S$(1)
50 FOR I=1 TO 32
60 E$=E$+CHR$(207)
70 NEXT I
79 REM ESQUIADOR HACIA LA IZQUIERDA
80 S$(0)=CHR$(201)
89 REM ESQUIADOR HACIA LA DERECHA
90 S$(1)=CHR$(198)
99 REM FONDO BLANCO
100 CLS 5
108 REM DIRECCION INICIAL:
109 REM                IZQUIERDA
110 D=-1
118 REM POSICION INICIAL
119 REM DEL ESQUIADOR
120 J=16
129 REM DISEÑO DE LAS PUERTAS
130 P$=CHR$(181)+CHR$(207)+CHR$(207)+CHR$(170)
134 REM
135 REM BUCLE PRINCIPAL
136 REM
140 FOR K=1 TO 300
145 REM CALCULO DE COORDENADAS
146 REM DEL ESQUIADOR
150 Y=INT(J/32)*2
160 X=(J-16*Y)*2
165 REM ESQUIADOR A LA ALTURA DE
166 REM UNA PUERTA?
170 IF K>=16 AND (K-5)/10=INT((K-5)/10) THEN
    GOSUB 330
179 REM FIJACION DE UNA PUERTA?
180 IF K<284 AND K/10=INT(K/10) THEN GOSUB
    350
189 REM MOVIMIENTO DEL ESQUIADOR
190 IF INKEY$<>"" THEN D=-D
200 J=J+D
210 IF J<2 THEN J=2
220 IF J>29 THEN J=29
230 PRINT @ 511,E$;
240 PRINT @ J,S$(D/2+0.5);
250 NEXT K
254 REM
255 REM FIN DEL RECORRIDO
256 REM
260 PRINT @ 164,"PUERTA(S) SALTADA(S) :";T;
270 PRINT @ 229,"OTRA BAJADA ?";
280 D$=INKEY$
290 IF D$="" THEN 280
300 IF D$<>"N" THEN RUN
310 CLS
320 END
324 REM
325 REM PUERTA SALTADA?
326 REM
330 IF POINT (X-2,Y)<>0 OR POINT (X+4,Y)<>3
    THEN IF POINT (X-4,Y)<>0 OR POINT
    (X+2,Y)<>3 THEN T=T+1:SOUND 1,1
340 RETURN
344 REM
345 REM FIJACION DE UNA PUERTA
346 REM
350 P1=RND(3)-2
360 P=P-6*P1
370 IF P<482 THEN P=488
380 IF P>506 THEN P=500
390 PRINT @ P,P$;
400 RETURN
```

Un robot en casa

Poco a poco están empezando a aparecer en el mercado robots de precio económico: el Beasty ha sido uno de los primeros



Cortesía de Commotion Ltd

El Beasty se suministra en forma de *kit*, y el usuario lo puede ensamblar con la ayuda de un par de destornilladores. Con el equipo también se suministra el software basado en cassette que contiene el sistema operativo Robol que se utiliza para controlar el brazo-robot.

Se proporcionan dos manuales. El folleto de construcción comienza con una larga introducción que versa sobre la historia de la robótica, para pasar luego a los detalles relativos a la construcción. Al usuario novato tal vez la gran cantidad de componentes y las un tanto complejas instrucciones le resulten algo intimidantes; se ofrecen ilustraciones, que pueden ser de mayor utilidad. No obstante, incluso el principiante será capaz de montar el brazo correctamente, si bien conseguirlo le llevará un poco de tiempo. De momento, el Beasty no se puede adquirir ya montado, si bien el fabricante, Commotion, afirma que lo puede ofrecer armado si así se le solicita.

Una vez montado, el Beasty se compone de una base que soporta una juntura que posibilita el movimiento lateral. Esta juntura lleva conectada una corta varilla de aluminio, que está unida a la parte superior del brazo mediante una segunda juntura. Una tercera juntura conecta el antebrazo. Estas junturas están alimentadas por servomotores, cada uno de los cuales controla dos cortas varillas rígidas que están conectadas al "esqueleto" del brazo. Cuando un servomotor gira, tira hacia él de una de las varillas y empuja a la otra en la dirección contraria, haciendo girar, por consiguiente, la juntura y moviendo el brazo. Un servomotor opera traduciendo impulsos digitales en movimiento. El motor recibe una serie de impulsos a una determinada frecuencia y el procesador interpreta estos impulsos como un ángulo de movimiento. Mientras la frecuencia permanezca constante, el motor mantendrá el brazo en su posición actual; un cambio en la frecuencia de los impulsos le indica al procesador que se requiere un nuevo ángulo y entonces el brazo se moverá.

Los servos FP128 utilizados en el Beasty pueden generar 3,5 kg/cm de empuje. Esto significa que, en 1 cm a lo largo de un eje, el servo puede levantar 3,5 k, mientras que lo largo de 10 cm puede levantar 350 g. Éste es un punto importante a tener en cuenta cuando se levantan pesos; obviamente, el servo del "hombro", al estar más alejado del peso a levantar, será el que soportará la mayor tensión.

El servoprocesador está alojado en una pequeña caja negra que no se halla unida al brazo propiamente dicho. Esta caja posee conectores para conectar hasta cuatro servomotores (la cuarta conexión es para un motor opcional que se puede utilizar para operar un "garfio" o cualquier dispositivo de agarre similar en el extremo del antebrazo). También hay un conector de entrada que se conecta en interface con la puerta para el usuario del BBC, un cable de potencia que se enchufa en el conector de potencia auxiliar del ordenador.

Una vez cargado el software desde cassette, la pantalla visualiza una indicación que le recuerda al usuario que el sistema está en modalidad Edit. En Robol, una línea de programa se compone de un número de línea, una construcción y una serie de números, cada uno de los cuales corresponde a una de las cuatro opciones de servomotores. Si la línea contiene la instrucción MOVE (mover), los números corresponden a la frecuencia de impulsos que man-

Ian McKinnell

Servomotores
El Beasty estándar viene equipado con tres servomotores conectados a una caja interface. Los servomotores son motores muy utilizados en robótica debido a la forma en que proporcionan una fuerza constante o "momento de torsión" correspondiente a la frecuencia de la señal digital suministrada

tienen a los servos en sus posiciones en curso. El usuario puede alterar estos números y mientras el sistema esté en modalidad Edit el servomotor que se esté regulando se moverá a medida que se efectúen las modificaciones, lo que permite posicionar el brazo tal como lo desee el usuario. Cuando el operador se sienta satisfecho respecto a las posiciones de cada uno de los motores, se debe pulsar Return, después de lo cual se visualiza una nueva línea en Robol y se puede programar la siguiente serie de movimientos.

Si se pulsa la tecla de función F0 se puede hacer que el brazo lleve a cabo una secuencia completa de movimientos. El programa se puede comenzar a ejecutar a partir de cualquier línea pulsando primero F1 y después F0 (ésta produce la ejecución desde el principio del programa), o bien cambiando el número de línea en curso mediante el empleo de las teclas para el cursor. Al final del programa la secuencia de acciones se repetirá de forma automática. Si el usuario deseara detener el programa, deberá cambiar una instrucción MOVE por STOP.

Demoras cronometradas

Durante la ejecución de una serie de instrucciones MOVE se puede hacer que el brazo haga una pausa mediante la incorporación de una instrucción WAIT (esperar), seguida de un número. Ésta trabaja accediendo a la patilla TIMER 1 de la puerta para el usuario, generando una interrupción. Dado que el reloj trabaja en unidades de 1/100 (centésimas) de segundo, WAIT 100 producirá una demora de 1 s antes de que se lleve a cabo la siguiente instrucción.

La acción del brazo se puede acelerar enormemente cambiando la instrucción MOVE por JUMP (saltar). También se incluyen dos sentencias de cronometraje: JDELAY y MDELAY. El Beasty posee una demora incorporada que se produce antes de la ejecución de cada línea. Ésta posee un valor por defecto de 20 (es decir, 1/5 s), pero este valor se puede alterar utilizando JDELAY para sentencias JUMP y MDELAY para instrucciones MOVE.

El sistema operativo Robol es fácil de utilizar, y programar el brazo para que realice movimientos complejos es una tarea sencilla que ocupa sólo minutos. El manual de operaciones es breve pero perfectamente adecuado, aunque los programadores avanzados quizá encuentren que la información que proporciona para una programación más compleja es insuficiente. El Beasty se puede controlar desde BASIC utilizando el programa *Driver*; éste accede a la puerta para el usuario del BBC Micro de forma muy similar a los ejemplos que hemos ofrecido en nuestro apartado de *Bricolaje*.

Commotion ha incluido, asimismo, un corto programa que permite efectuar copias del software Robol. Lamentablemente, la mayoría de las unidades de disco BBC emplean el conector de potencia auxiliar del BBC Micro, por lo cual no se pueden conectar al mismo tiempo el Beasty y una unidad de disco.

No obstante, a pesar de estos nimios detalles el Beasty es, ciertamente, una valiosa introducción al campo de la robótica. Se podría decir que éste es un dispositivo en espera de una aplicación, dado que en realidad no se puede afirmar que el brazo-robot sea auténticamente útil y es bastante probable que sólo lo adquieran los aficionados más entusiastas.

EL BEASTY

SOFTWARE
Robol y rutinas activadoras en cassette.

DOCUMENTACION
El Beasty viene con un manual de montaje y una guía de programación.

VENTAJAS
Constituye una introducción a la robótica fácil de utilizar y de precio económico.

DESVENTAJAS
La guía de montaje adolece de falta de claridad y existe una carencia de aplicaciones obvias para el brazo, aunque Commotion ha prometido algunas mejoras que ampliarán sus usos.

Por el laberinto

Ahora desarrollaremos un programa «inteligente» que conducirá a través de un laberinto el vehículo que hemos creado

La primera etapa en la construcción de un laberinto es decidir dónde lo realizaremos. Podría ser en la superficie de una mesa o sobre el suelo. El área designada se dividirá entonces en cierto número de cuadrados, dependiendo el tamaño de cada uno de éstos de las dimensiones del vehículo que se utilizará para cruzar el laberinto. Cada cuadrado debe ser lo suficientemente grande como para permitir que el vehículo pivote 360° en el interior de un único cuadrado. Entonces se puede marcar la superficie a modo de cuadrícula. Para conformar el laberinto se pueden colocar sobre la superficie objetos tales como libros, tazas o pequeños trozos de madera.

El programa exige que se especifiquen las dimensiones del laberinto y las posiciones de los cuadrados que estén ocupadas y aquellas que estén libres. El método más sencillo de hacer esto consiste en utilizar un código binario: 1 para indicar que un cuadrado está parcial o totalmente ocupado por un objeto, 0 para señalar que está vacío. Para que no sea necesario entrar los datos relativos al laberinto cada vez que se ejecute el programa, esta información se debe escribir como una serie de sentencias DATA. Los cuatro datos finales son las coordenadas del punto de partida y del punto de llegada. Podemos imaginar que el origen de coordenadas está en la esquina superior izquierda, siendo la fila superior la fila 0, y siendo la columna situada más a la izquierda la columna 0. Este laberinto es el correspondiente a las siguientes sentencias DATA:

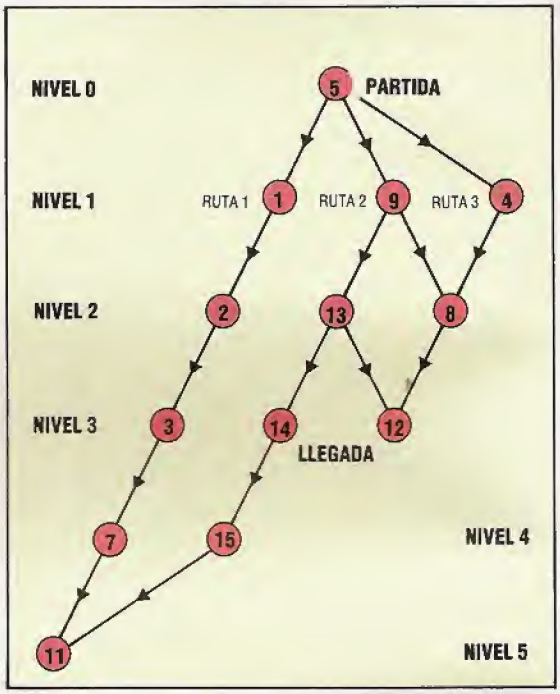
```
DATA 4,4:REM DIMENSIONES DEL LABERINTO
DATA 1,0,0,0,0,0,1,0
DATA 0,0,1,0,0,0,0,0
DATA 1,1:REM COORDS DE PARTIDA
DATA 2,3:REM COORDS DE LLEGADA
```

Encontrar una ruta a través del laberinto no presenta excesivas dificultades. Podemos diseñar un programa que trace un camino desde el punto de partida, retrocediendo en los callejones sin salida y volviendo hacia atrás los pasos necesarios hasta llegar finalmente al punto de llegada. La ruta hallada (sin los rodeos de los callejones sin salida) puede o no ser la ruta más corta posible. Si deseáramos hallar la *mejor* ruta entre los puntos de partida y de llegada, entonces deberíamos adoptar un procedimiento que pruebe todos los posibles caminos entre los dos puntos. Vale la pena señalar que nuestro programa interpreta que la "mejor ruta" es la que utiliza la menor cantidad de cuadrados.

Podemos simplificar la tarea de probar cada una de las rutas creando una estructura en los datos del laberinto que represente las relaciones existentes entre los cuadrados. La estructura de datos que mejor se presta para esta aplicación es un árbol jerárquico. Empezando con el punto de partida como la "raíz" del árbol, podemos construir una segunda generación de cuadrados (o "nudos") que están a una distancia de un cuadrado de la raíz. A partir de esta segunda generación de nudos se puede construir una tercera generación, y así sucesivamente. Podemos dibujar un árbol para cualquier laberinto numerando cada cuadrado y siguiendo la regla de que los descendientes de cualquier nudo se dibujan de izquierda a derecha por el orden Norte, Este, Sur y Oeste del nudo principal del laberinto.

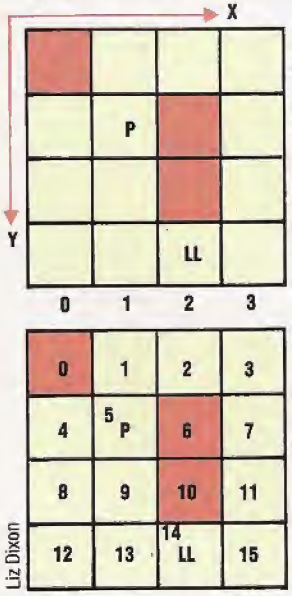
Este sencillo laberinto se puede resolver de cinco formas, sin desandar los pasos dados. En la ilustración superior se muestran tres posibles soluciones, como rutas a través del árbol y como rutas verdaderas por el laberinto. Para nosotros es evidente que la ruta 2 es la más corta, pero ello se debe a que podemos evaluar el árbol lateralmente, es decir, podemos considerarlo como un todo. El ordenador debe resolver el árbol de forma lineal, tomando sistemáticamente cada uno de los caminos posibles hasta hallar el nudo final o llegar a un callejón sin salida. En el primer caso se debe llevar un registro de la ruta de éxito; en el segundo, antes de volver a empezar desde el nudo raíz se debe señalar el camino tomado como un callejón sin salida. El programa continuará recorriendo el árbol hasta haber probado todas las ramas que parten del nudo raíz.

El BASIC no se presta fácilmente para tratar algoritmos de este tipo y con frecuencia la programación puede resultar torpe y difícil de manejar. Len-



Estructura de árbol
Para encontrar la mejor ruta a través del laberinto se debe construir un "árbol" que represente las relaciones entre los cuadrados. Cada nudo se considera de uno en uno, creando niveles de nudos inferiores. Los nudos del nivel 1 están a un cuadrado de distancia de la partida; los nudos del nivel 2 están a dos cuadrados de distancia, etc. Es bastante directo dibujar el árbol, pero implementar esta estructura en BASIC es más difícil

Conducción en doble sentido
Este programa para resolver un laberinto interpreta a éste de dos maneras. Dado que el laberinto se lee a partir de sentencias de datos, está almacenado en una matriz bidimensional, reteniéndose asimismo inicialmente los puntos de partida y de llegada como coordenadas. Con el fin de resolver el laberinto, el programa debe tratar cada uno de los cuadrados del laberinto como el "nudo" de un árbol. En vez de utilizar el sistema de coordenadas inicial, cada cuadrado se numera, por consiguiente, por orden, empezando en el rincón superior izquierdo del laberinto



Liz Dixon

guajes como el LOGO y el ALGOL están mucho mejor dotados para cumplir esta misión. En BASIC tenemos que llevar a cabo dos tareas fundamentales: en primer lugar, debemos obtener nuestro árbol a partir de los datos del laberinto, tal como se le presentan al programa. Y por cada cuadrado del laberinto debemos tener cuatro apuntadores que indiquen qué cuadrado hay en cada una de las cuatro direcciones. Para almacenar este sistema de señaladores, lo más indicado es una matriz bidimensional, TR(N,D), donde N es el número del cuadrado y D es la dirección de 1 a 4. Por consiguiente, en nuestro laberinto simple, TR(9,1) sería 5: el cuadrado que se halla al norte del cuadrado 9. Cuando el cuadrado de una dirección determinada no está libre, o si hay una frontera del laberinto, ello se puede marcar mediante un valor especial, por ejemplo -1.

A medida que se va recorriendo el árbol, la ruta tomada se almacena en una seudopila, implementada utilizando una matriz unidimensional y una variable, D, para indicar el siguiente espacio disponible en la pila. El camino más corto que se encuentre también se almacena en una matriz unidimensional, con el número de pasos para la ruta almacenado en el primer elemento de la matriz.

Cuando el programa ha recorrido su camino a través del árbol, se retendrá un registro de la mejor ruta en forma de una serie de números de cuadrados. Suponiendo que originalmente el vehículo estuviera orientado hacia el Norte en el cuadrado de partida, se lo puede dirigir utilizando las relaciones matemáticas simples entre la dirección a recorrer y la diferencia entre dos números de cuadrados consecutivos en la matriz de ruta. Por ejemplo, en nuestro sencillo laberinto, una diferencia de +4 indicaría norte, −4 indicaría sur, y así sucesivamente. Debemos entonces calcular el ángulo a describir para cambiar la dirección, antes de seguir adelante hacia otro cuadrado. Dado que el vehículo emplea motores eléctricos CD simples, los ángulos de giro y las distancias recorridas se gobiernan mediante el período de tiempo durante el cual está encendida una determinada combinación de motores. Para hacer un uso práctico del programa es necesario realizar algunos experimentos iniciales para determinar los intervalos de tiempo requeridos para describir un ángulo de 90° y avanzar un cuadrado. Esta información se debe entrar en las variables AF y FF, respectivamente. La versión para el BBC exige unidades de una centésima de segundo, y la versión para el Commodore 64 requiere unidades de un sesentavo de segundo.

Ian McKinnell

Solución al laberinto

```
5700 FOR P=0 TO SX*SY-1:FOR I=1 TO 4:TR(P,I)=-1:NEXT I,P
5710 :
5720 REM ** CALCULAR PARTIDA Y LLEGADA ****
5730 X=XS:Y=YS:GOSUB 9200:S=N
5740 X=XF:Y=YF:GOSUB 9200:F=N
5750 :
5760 REM ** CONSTRUIR **
5770 LC=1:CC=0:REM INIC APUNTADORES PILA
5780 LN(LC)=S:REM PUNTO PARTIDA
5790 CN=LN(LC):REM TOMAR DE LA PILA NUDO EN CURSO
5800 DF=0
5810 FOR D=1 TO 4
5820 N=CN+DR(D):GOSUB 8900:REM CONVERTIR A X,Y
5830 IF(X<0 OR X>SX-1 OR Y<0 OR Y>SY-1)THEN 7300
5840 IF MZ(X,Y)=1 THEN 7300
5850 IF (D=2 AND N/SX=INT(N/SX))THEN7300
5860 IF (D=4 AND CN/SX=INT(CN/SX))THEN7300
5870 IF TR(N,ID(D))=CN THEN 7300
5880 TR(CN,D)=N:DF=1
5890 CC=CC+1:CN(CC)=N:REM PONER EN LA PILA
5900 NEXT D
5910 IF(DF=0 AND LC=1) THEN RETURN: REM NUDO TERMINAL
5920 :
5930 LC=LC-1:REM DECREMENTAR ULTIMO APUNT PILA
5940 IF LC>0 THEN 6450:REM PROXIMO NUDO
5945 :
5950 REM ** COPIAR PILA EN CURSO EN ULTIMA PILA **
5960 FOR I=1 TO CC:LN(I)=CN(I):NEXT I
5970 LC=CC:CC=0:GOTO6450:REM SIGUIENTE NUDO
5980 :
5990 REM **** RECORRER ARBOL ****
6000 C=0:RN=S:CN=RN:EF=0
6010 D=C+1:RD(C)=CN
6020 IF CN=F THEN GOSUB 8100:GOSUB8000:IF EF=0 THEN 7730
6030 IF EF=1 THEN RETURN
6040 DF=0
6050 FOR D=1 TO 4
6060 IF TR(CN,D)<>-1 THEN CN=TR(CN,D):DF=1:DR=D:D=4
6070 NEXT D
6080 IF DF=0 THEN GOSUB 8000
6090 IF EF=0 THEN 7730
6100 IF EF=1 THEN RETURN
6110 :
6120 REM *** RECOMENZAR POR LA RAIZ ****
6130 TR(RT(C-1),DR)=-1
6140 CN=RN:C=0
6150 IF(TR(CN,1)AND TR(CN,2)AND TR(CN,3)AND TR(CN,4)=-1 THEN EF=1
6160 RETURN
6170 :
6180 REM **** GUARDAR MATRIZ ****
6190 IF C>SR(0) THEN RETURN:REM ES MAS CORTA LA RUTA NUEVA?
6200 SR(0)=C
6210 FOR I=1 TO C:SR(I)=RT(I):NEXT I:RETURN
6220 :
6230 REM **** GUIAR VEHICULO ****
6240 PD=1:REM SUPONER DIRECCION INICIAL ES NORTE
6250 FOR C=1 TO SR(0)-1
6260 DF=SR(C+1)-SR(C)
6270 :
6280 REM ** HALLAR DIRECCION REQUERIDA **
6290 FOR I=1 TO 4
6300 IF DF=DR(I) THEN D=I:I=4
6310 NEXT I
6320 :
6330 DR=D-PD:PD=D
6340 H=INT(4+DR/4):R=(4+DR)-4*H
6350 :
6360 REM ** EFECTUAR GIRO **
6370 FOR I=1 TO R
6380 POKE REGDAT=9:REM GIRO EN SENTIDO HORARIO
6390 T=TI
6400 IF (TI-T)<AF THEN 8310:REM ESPERAR
6410 POKE REGDAT=0:REM APAGAR
6420 NEXT I
6430 :
6440 REM *** ADELANTE ***
6450 POKE REGDAT=5
6460 T=TI
6470 IF (TI-T)<FF THEN 8380
6480 POKE REGDAT=0
6490 NEXT C
6500 :
6510 RETURN
6520 :
8900 REM **** CONVERTIR N A X,Y ****
9000 Y=INT(N/SX):X=N-SX*Y: RETURN
9200 :
9210 REM **** CONVERTIR X,Y A N ****
9220 N=Y*SX+X:RETURN
```

Para el BBC

Efectuar las siguientes modificaciones:

```
3130 RDD+&FE62:REGDAT=&FE60
8290 ?REGDAT=9
8300 TIME=0
8310 REPEAT UNTIL TIME>=AF
8320 ?REGDAT=0
8360 ?REGDAT=5
8370 TIME=0
8380 REPEAT UNTIL TIME>=FF
8390 ?REGDAT=0
```

El juego de la espada

Vamos a crear un juego de aventuras basado en texto. Lo haremos mediante un enfoque general para que usted pueda construir su propio juego

En este capítulo nos limitaremos a analizar los aspectos más generales de la programación de un juego de aventuras, y en el próximo consideraremos los detalles específicos para un juego en particular.

En todos los juegos de aventuras hay cinco actividades básicas que el jugador debe ser capaz de llevar a cabo: se necesita recoger objetos o abandonarlos, listar las cosas que uno lleva consigo, observar los alrededores y desplazarse por el juego de un cuarto a otro (o de escenario en escenario). De modo que son éstas las instrucciones básicas que vamos a programar antes que nada. Por razones de simplicidad, limitaremos la forma de las instrucciones a uno de dos tipos: o palabras individuales (como MIRAR) o bien pares compuestos por verbo y sustantivo (como ARROJAR ANILLO). El programa llevará dos listas: una denominada INVENTARIO, que incluirá todo lo que el jugador lleve consigo en cada momento, y la otra, llamada CONTENIDO, será una relación de los objetos existentes en el cuarto actual.

La primera instrucción que vamos a definir es INVENTARIO:

```
TO INV
  PRINT [UD. LLEVA:]
  IF EMPTY? :INVENTARIO THEN PRINT [NADA]
  ELSE PRINT :INVENTARIO
END
```

Observe que este procedimiento utiliza la forma completa de la sentencia IF: IF ‹condición› THEN ‹acción 1› ELSE ‹acción 2›. La instrucción para coger un objeto será COGER:

```
TO COGER :ITEM
  IF MEMBER? :ITEM :CONTENIDO
  THEN COGERLO :ITEM ELSE PRINT [NO
  PUEDO NO ESTA AQUI]
END
```

MEMBER? es una primitiva que verifica si un elemento pertenece a una lista. Para "coger" un artículo necesitamos hacer dos cosas: añadirlo al inventario y eliminarlo de la lista de contenido. Éstos son los procedimientos que realizan estas tareas:

```
TO COGERLO :ITEM
  AGREGAR.EN.INV :ITEM
  SACAR.DEL.CUARTO :ITEM
END

TO AGREGAR.EN.INV :ITEM
  MAKE"INVENTARIO SENTENCE :ITEM
  :INVENTARIO
END

TO SACAR.DEL.CUARTO :ITEM
  MAKE"CONTENIDO BORRAR :ITEM
  :CONTENIDO
END
```

El último de estos procedimientos implica borrar un elemento de una lista, tarea que fue uno de los ejercicios sugeridos en el capítulo anterior.

```
TO BORRAR :ITEM :LISTA
  IF:ITEM=FIRST :LISTA THEN OUTPUT BUTFIRST
  :LISTA
  OUTPUT SENTENCE FIRST :LISTA BORRAR :ITEM
  BUTFIRST :LISTA
END
```

La instrucción para dejar un objeto se implementa de forma similar:

```
TO DEJAR :ITEM
  IF MEMBER? :ITEM :INVENTARIO THEN
  DEJARLO :ITEM ELSE PRINT [NO PUEDES
  DEJARLO PORQUE NO LO TIENES!]
END

TO DEJARLO :ITEM
  SACAR.DEL.INV :ITEM
  AGREGAR.AL.CUARTO :ITEM
END

TO SACAR.DEL.INV :ITEM
  MAKE"INVENTARIO BORRAR :ITEM :INVENTARIO
END

TO AGREGAR.AL.CUARTO :ITEM
  MAKE"CONTENIDO FPUT :ITEM :CONTENIDO
END
```

Habiendo entrado todos los procedimientos que hemos dado hasta ahora, es el momento de verificar su funcionamiento. En primer lugar, debemos definir las dos variables globales (INVENTARIO y CONTENIDO) y después comprobarlas para las siguientes instrucciones:

```
MAKE"CONTENIDO [ESPADA LANZA ANTORCHA]
MAKE"INVENTARIO [FAROL]
COGER "ESPADA
DEJAR "FAROL
```

Ahora examine CONTENIDO e INVENTARIO utilizando estas sentencias:

```
PRINT :CONTENIDO
PRINT :INVENTARIO
```

y compruebe que estén correctas.

Observe que hemos empleado comillas antes de los nombres de los objetos tanto en la instrucción COGER como en DEJAR. El empleo de las comillas de esta forma ya es instintivo para el programador

de LOGO, pero es probable que le resulte muy confuso a un "aventurero" que no sepa nada acerca del lenguaje. Para posibilitar el empleo de la forma COGER ESPADA, que es más natural, debemos definir ESPADA de la siguiente manera:

```
TO ESPADA
  OP"ESPADA
END
```

Por supuesto, tendremos que hacer lo mismo con cada sustantivo que se utilice en nuestro juego de aventuras basado en texto.

La instrucción MIRAR imprimirá una descripción del cuarto en curso, una lista de lo que contenga y las posibles rutas de salida del cuarto. Para hacer esto necesitaremos otras dos listas: una primera de descripciones y una segunda de salidas. Con el objeto de poder incluir descripciones extensas que ocupen más de una línea de la pantalla, la lista de descripciones se define como una lista de listas. Por ejemplo:

```
MAKE"DESCRIPCION [[UD ESTA PARADO JUNTO
  A LA ENTRADA][DE UNA CUEVA]]
```

Para llevar un registro de cómo se unen los cuartos entre sí, cada cuarto tiene asignado un número. La lista de salidas es simplemente una lista de sublistas, cada una de ellas compuesta por una dirección y un número de cuarto.

Por tanto:

```
MAKE"LISTA.SALIDAS [[N 4][E 6]]
```

Ahora ya podemos definir MIRAR:

```
TO MIRAR
  PRINTL :DESCRIPCION
  PRINT "
  PRINT [UD VE:]
  IF EMPTY? :CONTENIDO THEN PRINT [NADA
  ESPECIAL]ELSE PRINT :CONTENIDO
  PRINT "
  PRINT [PUEDE IR:] IMPRIMIR. SALIDAS
   LISTA.SALIDAS PRINT "
END
```

En este procedimiento se han empleado dos rutinas de impresión especiales para que la visualización resulte más fácil de leer. Se utiliza PRINTL para imprimir varias líneas de texto.

```
TO PRINTL :LISTA
  IF EMPTY? LISTA THEN STOP
  PRINT FIRST :LISTA
  PRINTL BUTFIRST LISTA
END
```

IMPRIMIR.SALIDAS imprime las salidas del cuarto sin imprimir los números de los cuartos.

```
TO IMPRIMIR.SALIDAS :LISTA
  IF EMPTY? :LISTA THEN PRINT "STOP
  MAKE "SALIDA FIRST :LISTA
  PRINT1 FIRST :SALIDA
  PRINT1 " "
  IMPRIMIR.SALIDAS BUTFIRST :LISTA
END
```

Podemos describir todo cuanto se sepa acerca de un cuarto del juego poniendo juntas las tres sublistas: la descripción, el contenido y las salidas. Por ejemplo:

```
MAKE "CUARTO.1 [[[UD ESTA PARADO JUNTO A LA
  ENTRADA] [A UNA CUEVA]] [ESPADA] [[N 4][E 6]]]
```

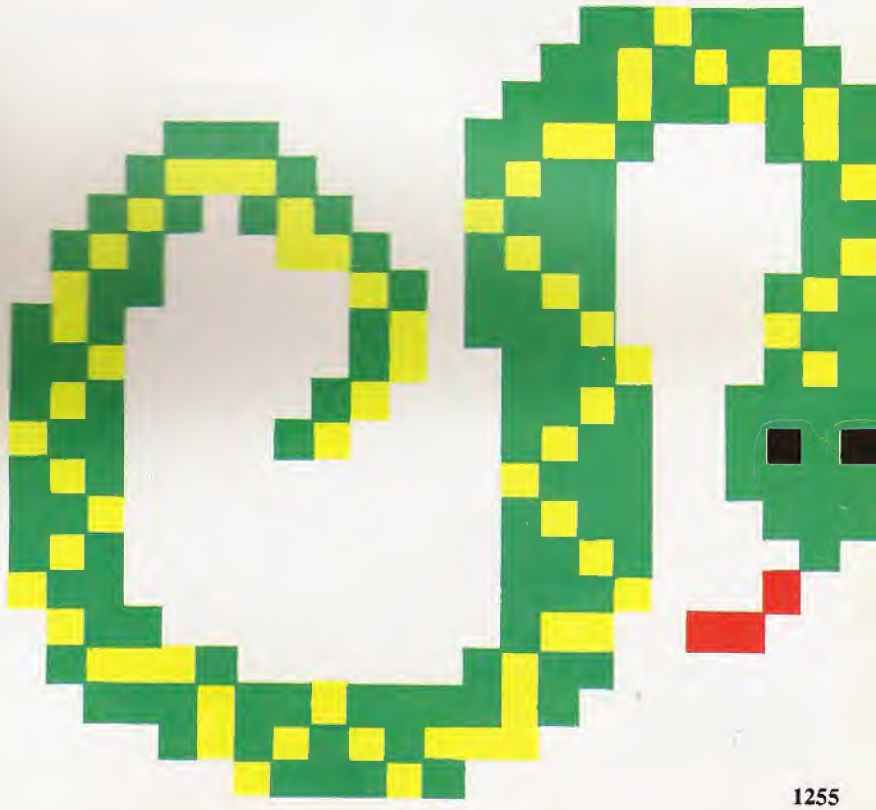
Dado que CUARTO.1 se define de esta forma, podríamos dividirla en sus componentes individuales con el siguiente procedimiento:

```
TO ASIGNAR.VARIABLES
  MAKE "CUARTO THING "CUARTO.1
  MAKE "DESCRIPCION DESCRIPCION :CUARTO
  MAKE "CONTENIDO CONTENIDO :CUARTO
  MAKE "LISTA.SALIDAS LISTA.SALIDAS :CUARTO
END
```

THING "CUARTO.1 es una alternativa de :CUARTO.1; significa "el contenido de la variable CUARTO.1". Luego explicaremos el motivo de la utilización de esta forma. Los subprocedimientos se definen de esta manera:

```
TO DESCRIPCION :CUARTO
  OUTPUT ITEM 1 :CUARTO
END

TO CONTENIDO :CUARTO
  OUTPUT ITEM 2 :CUARTO
END

TO LISTA.SALIDAS :CUARTO
  OUTPUT ITEM 3 :CUARTO
END
```

Tal como está, este procedimiento sólo funciona para CUARTO.1. Necesitamos ampliarlo de modo que se lo pueda utilizar con un carácter más general para cualquier cuarto. Esto lo hacemos empleando una variable global, AQUI, que contiene el número del cuarto en curso. Supongamos que ahora éste sea 2. La primitiva WORD del LOGO produce una palabra compuesta por una combinación de sus dos entradas (por tanto, WORD "CUARTO.:AQUI produciría CUARTO.1). Entonces le asignamos este nombre a la variable NOMBRE.CUARTO; en consecuencia, NOMBRE.CUARTO es CUARTO.2. Ahora podemos asignar



```
CUARTO como THING :NOMBRE.CUARTO.
  TO ASIGNAR.VARIABLES
  MAKE "NOMBRE.CUARTO WORD "CUARTO.:AQUI
  MAKE "CUARTO THING :NOMBRE.CUARTO
  MAKE "DESCRIPCION DESCRIPCION :CUARTO
  MAKE "CONTENIDO CONTENIDO :CUARTO
  MAKE "LISTA.SALIDAS LISTA.SALIDAS :CUARTO
END
```

Ahora ya está en condiciones de dibujar un mapa de los escenarios de su juego de aventuras y listar las descripciones de los mismos (con contenido y salidas). En el próximo capítulo acabaremos nuestro análisis general estudiando el movimiento entre escenarios y cómo se implementan los "peligros". Después comenzaremos a considerar un juego de aventuras completo como ejemplo de lo que se puede hacer.

Complementos al LOGO

Algunas versiones de LOGO MIT no poseen EMPTY?, ITEM, COUNT ni MEMBER? En el capítulo anterior ofrecimos definiciones para las tres primeras. La definición de MEMBER? es:

```
TO MEMBER? :ITEM :LISTA
  IF :LISTA = [] THEN OUTPUT "FALSE
  IF :ITEM = FIRST :LISTA THEN OUTPUT "TRUE
  OUTPUT MEMBER? :ITEM BUTFIRST :LISTA
END
```

En todas las versiones LCSI utilice:

```
EMPTYP por EMPTY?
LISTP por LIST?
MEMBERP por MEMBER?
TYPE por PRINT1
```

También hay una primitiva, EQUALP, que comprueba si sus dos entradas son la misma. Utilícela para comparar listas y palabras en lugar de =. (El signo igual funciona para listas sólo en algunas versiones LCSI.) La sintaxis de IF completa en LOGO LCSI se demuestra mediante este ejemplo:

```
IF EMPTYP :CONTENIDO [PRINT [NADA ESPECIAL]][PRINT :CONTENIDO]
```

Si la condición es verdadera se lleva a cabo la primera lista después de la condición, y si es falsa, se lleva a cabo la segunda.

Respuestas

1. Para imprimir en orden inverso:

```
TO PRINTR :LISTA
  IF EMPTY? :LISTA THEN PRINT "STOP
  PRINT1 LAST :LIST
  PRINT1" "
  PRINTR BUTLAST :LISTA
END
```

El siguiente procedimiento produce la lista en orden inverso, en lugar de imprimirla:

```
TO INVERTIR :LISTA
  IF EMPTY? :LISTA THEN OUTPUT []
  OUTPUT SENTENCE LAST :LISTA INVERTIR
  BUTLAST :LISTA
END
```

2. En el texto principal de este capítulo se incluye un procedimiento para borrar un elemento de una lista.

Poesía LOGO

En el capítulo anterior le planteamos el problema de mejorar las aptitudes del LOGO para escribir poesía. Un posible método consiste en crear un "modelo" de una estructura de frase (tal como "sustantivo, verbo, sustantivo") y elegir palabras de sublistas de estas partes de la frase. Por ejemplo, utilizando este modelo:

```
POEMA2 [SUSTANTIVO VERBO SUSTANTIVO]
```

podríamos tener:

```
TO POEMA2 :MODELO
  IF EMPTY? :MODELO PRINT "STOP
  POEMA2.1 FIRST :MODELO
  POEMA2 BUTFIRST :MODELO
END

TO POEMA2.1 :PAL
  IF :PAL="SUSTANTIVO (PRINT1" ' '
  GETRANDOM :SUSTANTIVO)
  IF :PAL="VERBO (PRINT2" ' '
  GETRANDOM :VERBO)
END
```

Este método sería muy incómodo si introdujéramos muchas más partes de la oración. Analicemos las variables más atentamente para ver si podemos mejorar algo. En primer lugar, entre:

```
MAKE "ROSA "DULCE
MAKE "OTRONOMBRE "ROSA
```

Ahora descubrirá que:

```
PRINT :ROSA imprime DULCE
PRINT :OTRONOMBRE imprime ROSA
PRINT THING :OTRONOMBRE imprime DULCE
```

THING nos da el valor asociado a un nombre. En el último caso, el nombre que sigue a THING es el valor de la variable OTRACOSA, o sea, ROSA. Utilizando esta idea, podemos reescribir nuestro procedimiento del poema:

```
TO POEMA2.1
  (PRINT1" ' 'GETRANDOM THING :PAL)
END
```

La llamada a procedimiento POEMA2.1 "SUSTANTIVO le asigna a la variable PAL el valor SUSTANTIVO. Entonces el valor asociado con SUSTANTIVO es THING :PAL, la lista de sustantivos. Utilizando el modelo:

```
POEMA2 [ART SUSTANTIVO ADJ VERB ADV
  PREP ART ADJ SUSTANTIVO]
```

junto con una adecuada selección de vocabulario, hemos obtenido la siguiente salida:

```
UN PLANETA VERDE GIRA RUIDOSAMENTE
  BAJO UNA PARANOICA CUPULA
UNA NAVE RUINOSA VOLABA LENTAMENTE
  HACIA EL ESPLENDIDO PLANETA
```

Errores no, gracias

Continuamos con la elaboración del programa depurador en lenguaje máquina iniciado en el capítulo anterior

Para el módulo de E/S hay que elaborar cuatro rutinas más: GETHX2, GETHX4, PUTHEX y PUTCR. Los dos primeros procesos sirven para tomar desde el teclado dígitos hexadecimales: GETHX2 se encarga de los números hexa de dos dígitos y GETHX4 de los de cuatro dígitos. Para diseñar estas rutinas, lo primero que hay que hacer es saber si vamos a solicitar la entrada siempre de dos o cuatro dígitos (lo cual es fácil de programar pero incómodo para el usuario) o bien aceptar un número de caracteres seguidos de un Return. El problema siguiente es decidir si se permitirá el carácter de retroceso para borrar los caracteres ya introducidos.

Vamos a adoptar el método más sencillo para la rutina GETHX4: sólo se dará entrada a cuatro dígitos y no se empleará el retroceso. El valor de 16 bits (que significa una dirección) puede retornarse en el registro D.

GETHX2 plantea más de un problema si consideramos las circunstancias en que será usado. Deberán entrarse cantidades de 8 bits para inspeccionar y cambiar la memoria (orden M), lo que significa acceder a una dirección. El contenido de ésta es visualizado, pudiendo a continuación el usuario pulsar Return (para pasar a la posición siguiente en secuencia) o bien un número hexadecimal de dos dígitos (para ser almacenado en esa posición) o incluso cualquier otro carácter (un punto, p. ej., para volver al nivel de órdenes). Podemos añadir los dos caracteres válidos para indicar el fin de una cadena de dígitos hexa. GETHX2 tendría, pues, que aceptar dos dígitos hexa, un Return o un punto. El valor de ocho bits se puede retornar en B y hay que emplear A para indicar en qué situación se está. Si lo que se ha introducido es un número de dos dígitos, A tendrá valor 0; si se trata de Return, tendrá valor 1, y si es un punto, valor −1. Tales valores no permiten comprobar el contenido de A sin compararlo con otro valor.

Supongamos de momento que para este módulo se han hecho las siguientes declaraciones:

```
HEXCHS    FCC'0123456789ABCDEF'
DOT       FCB '.'      (punto)
RETURN    FCB 13       (Return en código ASCII)
```

Podemos pasar 16 como la longitud de la cadena para GETHX4, donde sólo necesitamos dígitos hexa, y 18 para GETXH2, donde se necesitan también los otros dos caracteres.

Rutina GETHX2

Data:
Carácter-siguiente es el ASCII contenido en A
Desplazamiento a la tabla de Car-Válidos, está en B
Valor-hexa se construye en B y es un valor de 8 bits
Flag está en A y es 0,1 o bien −1

Proceso:
Tomar Carácter-Siguiente
IF carácter es un punto (Desplazamiento = 16) entonces
  Poner Flag a −1
ELSE si el carácter es un Return (Desplazamiento = 17) entonces
  Poner Flag a 1
ELSE
  Guardar Desplazamiento temporalmente
  Tomar Carácter-Siguiente (en este punto sólo son válidos dígitos hexas)
  Construir Valor-Hexa
ENDIF

La forma final codificada de GETHX2 la encontrará el lector en la página 1259. Por otro lado, la codificación de la rutina GETHX4 se hace ahora más fácil con algunas piezas de la presente rutina. Haciendo HX4 un punto alternativo de entrada a la rutina GETHX2, es posible llamar a esa rutina y asegurarnos de que sólo se aceptan dígitos hexa válidos, siempre que carguemos B con un 16 antes de la llamada. De esta manera el proceso de toma de cuatro dígitos hexa se simplifica considerablemente.

Rutina GETHX4

Data:
Número-hexa, es un valor de 16 bits que se retorna en D
Byte-Más-Significativo, y
Byte-Menos-Significativo, son dos valores de 8 bits que serán retornados en B

Proceso:
Tomar Byte-Más-Significativo
Guardar Byte-Más-Significativo temporalmente
Tomar Byte-Menos-Significativo
Construir Número-Hexa

Damos la codificación final de esta rutina después de la de GETHX2.

El diseño de las rutinas para la visualización de caracteres es mucho menos complicado. En la rutina PUTHEX, supondremos que el número de 8 bits que necesitamos se encuentra en B.

Rutina PUTHEX

Data:
Número, es el valor de 8 bits que se encuentra en B
Desplazamiento, es el desplazamiento de 4 bits colocado en HEXCHS

Proceso:
Extraer los 4 bits más significativos de Número para

emplearlos como Desplazamiento
Visualizar HEXCHS (Desplazamiento)
Extraer los 4 bits menos significativos de Número
Visualizar HEXCHS (Desplazamiento)

La rutina final necesaria para manejar las entradas y salidas es la subrutina PUTCR. Ésta es inmediata, y la codificación final se explica por sí misma. Una vez codificadas todas las rutinas que se necesitan, podemos ya diseñar el módulo entero de E/S.

Módulos de Entrada/Salida

Proceso:
Tomar Orden devolverá Desplazamiento en B, el cual puede servir de desplazamiento para una tabla de salto
Tomar Dirección deja la dirección de retorno en D
Tomar Valor deja el valor de retorno en B, flag en A
Visualizar Valor, pasado en B
Visualizar Dirección, pasado en D

La codificación final del módulo E/S se encuentra en la página siguiente. Ahora ya podemos volver al módulo de Puntos de Ruptura que iniciamos en el capítulo anterior (véase p. 1238). Ya dimos la codificación del segundo proceso de este módulo, que establece los puntos de ruptura. Pospusimos el problema de la codificación del primer proceso (insertar puntos de ruptura) porque presuponía la obtención de una dirección. Dado que esta tarea ya la hemos encarado en las rutinas que proporcionamos aquí, podemos seguir adelante ofreciendo la versión codificada del proceso, que incorpora una bifurcación a la subrutina GETADD.

Obsérvese que en el código, la instrucción INC NUMBP, PCR suma un 1 a Número-Puntos-Ruptura, que es el desplazamiento correcto en la tabla de puntos de ruptura. Sin embargo, la dirección se retorna en D, y esto destruirá el valor contenido en A,

La rutina GETHX2

```
GETHX2  LDB    # 18          Número-car-val
HX4     PSHS   X             Guarda el registro empleado
        LEAX   HEXCHS,PCR    Toma en X la dir. de Car-Válidos
        BSR    GETCH         Toma Carácter-Siguiente
IFOO    CMPB   # 16          Si Desplazamiento = 16
        LDA    # $FF          Poner el flag a -1 (en
        BRA    ENDFOO        complemento a dos)
        CMPB   # 17          Si Desplazamiento = 17
        LDA    # 1           Poner a 1 el flag
        BRA    ENDFOO
        LSLB                 Desplaza B cuatro lugares a la
        LSLB                 izquierda para formar el dígito
        LSLB                 más significativo; B conserva el
        LSLB                 desplazamiento en HEXCHS y por
                             ello el valor binario
        PSHS   B             Guarda B temporalmente
        LDB    # 16          Sólo son válidos dígitos hexa
        BSR    GETCH         Carácter-Siguiente
        ADDB   1,S+          Construye un número de 8 bits y
        PULS   X,PC          pierde B temporalmente
```

La rutina PUTCR

```
PUTCR   PSHS   A             Guarda A
        LDA    # 13          Return en código ASCII
        BSR    OUTCH         Lo visualiza
        PULS   A,PC
```

La rutina GETHX4

```
GETHX   LDB    # 16
        BSR    HX4           Toma Byte-Más-Significativo
        PSHS   B             Guarda Byte-Más-Significativo
        LDB    # 16          temporalmente
        BSR    HX4           Toma Byte-Menos-Significativo
                             en B
        PULS   A             Toma Byte-Más-Sign. para
                             devolverlo a A
        RTS                  El valor requerido está en D
```

Rutina Visualizar-Puntos-de-Ruptura

```
BPLABS  FCC    '1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16'
SPACE   FCB    32            Espacio en código ASCII
DISPBP  PSHS   A,B,X,Y
        LEAX   BPTAB,PCR     Dir. de Tabla-Puntos-Ruptura
        LEAY   BPLABS,PCR    Dir. de etiquetas
        CLRB                 Pone en despl. 0 a Núm.-Punto-
                             Rupt.
WHIL01  CMBP   NUMBP,PCR     While Número-Punto-Ruptura <
        BGT    ENDW01        = Número-Puntos-Ruptura
        LDA    ,Y+           Visualiza etiqueta
        BSR    OUTCH
        LDA    ,Y+
        BSR    OUTCH
        LDA    SPACE,PCR     Visualiza un Espacio
        BSR    OUTCH
        PSHS   B             Guarda temporalmente B
        LDD    ,X++          Guarda Dirección
        BSR    DSPADD
        PULS   B             Restaura B
        BRA    WHIL01
ENDW01  PULS   A,B,X,Y       Restaura y retorna
```

puesto que el registro D comprende los registros A y B. Ésta es la razón por la que empleamos Y para conservar la dirección actual.

Habiendo codificado el primer proceso del módulo de Puntos de Ruptura, quedan aún tres procesos más por codificar. Dos de los cuales invierten los dos procesos que hasta ahora hemos codificado: Eliminar-Punto-Ruptura quitará de la tabla un punto de ruptura, mientras que Desactivar-Punto-Ruptura quita el Opcode SWI y devuelve el valor original. En el próximo capítulo estudiaremos estas dos rutinas. La tercera, que visualiza todos los puntos de ruptura, es la última que codificaremos.

Visualizar-Puntos-De-Ruptura

Data:

Número-Punto-Ruptura, es un contador de 8 bits para recorrer la tabla de Puntos de Ruptura; se encuentra en B

Punto-Ruptura-Actual, es la dirección a visualizar

Etiquetas-de-Puntos-Ruptura, son números (decimales) de dos dígitos para etiquetar las direcciones conforme son visualizadas

Espacio, es el carácter espacio que separa una etiqueta de una dirección

Proceso 3: Visualizar-Puntos-de-Ruptura

```
Poner Número-Punto-Ruptura a 1 (un
  desplazamiento actual de cero)
While Número-Punto-Ruptura < = Número-
  Puntos-Ruptura
    Visualizar Etiquetas-de-Puntos-Ruptura
      (Número-Punto-Ruptura)
    Visualizar Tabla-Puntos-Ruptura (Número-
      Punto-Ruptura)
    Incrementar Número-Punto-Ruptura
Endwhile
```

Fin de Proceso 3

La rutina PUTHEX

```
PUTHEX  PSHS   A,B,X        Guarda regs. empleados
        PSHS   B            Guarda B temporalmente
        LEAX   HEXCHS,PCR   Dir. de HEXCHS en X
        LSRB                Desplazamiento de cuatro lugares
        LSRB                para los bits más significativos
        LSRB
        LSRB
        LDA    B,X          Toma el carácter adecuado y lo
                            coloca en A
        BSR    OUTCH        Lo visualiza
        PULS   B            Toma de nuevo B
        ANDB   #%00001111   Enmascara los cuatro bits más
                            significativos
        LDA    B,X          Segundo carácter
        BSR    OUTCH
        PULS   A,B,X,PC     Restaura y retorna
```

El módulo Entrada/Salida

```
HEXCHS  FCC    '0 1 2 3 4 5 6 7 8 9 A B C D E F'
DOT     FCB
RETURN  FCB    13           Return en código ASCII
COMNDS  FCC    'BUDSGRMQ'
GETCOM  PSHS   A,X          Guarda los contenidos de A y X
        LEAX   COMNDS,PCR   Coloca la dirección de los
                            caracteres de la orden en X
        LDB    # 8          Número-Car-Vál
        BSR    GETCH        Toma Carácter
        PULS   A,X,PC       Return
GETADD  BSR    GETHX4
        BSR    PUTCR
        RTS
GETVAL  BSR    GETHX2
        BSR    PUTCR
        RTS
DSPVAL  BSR    PUTHEX
        BSR    PUTCR
        RTS
DSPADD  PSHS   B            Guarda B temporalmente
        TFR    A,B          Byte-Más-Significativo en B
        BSR    PUTHEX
        PULS   B            Recupera B
        BSR    PUTHEX
        PULS   B
        BSR    PUTHEX
        BSR    PUTCR
        RTS
```

Rutina Inserción-Puntos-Ruptura

```
BPOI    PSHS   A,B,X,Y      Guarda regs. empleados
        LDX    BPTAB        Dirección de Tabla-Puntos-
        LDA    NUMBP,PCR    Ruptura
IF01    CMPA   MAXBP,PCR    Si Número-Puntos-Ruptura <
        BGE    ENDF01       Max
        INC    NUMBP,PCR    Sumar 1 a Número-De-Puntos
                            Ruptura
        LSLA                Multiplica por dos el
        LEAY   A,X          Desplazamiento para la tabla de
                            16 bits
        BSR    GETADD       Toma la dirección
        STD    ,Y           Almacena la dirección en Tabla-
                            Puntos-Ruptura
ENDF01  PULS   A,B,X,Y      Restaura y retorna
```

Fuera de la ley

En este juego el usuario personifica a Mugsy, jefe de una banda de gángsters en la Norteamérica de los años treinta

El papel de Mugsy consiste en tomar diversas decisiones sobre el número de "clientes" de la banda en los que se puede confiar, cuánto dinero se puede gastar en municiones para la banda, y las sumas que se le deben abonar a la policía en concepto de soborno. Si no se gasta el dinero suficiente en armas, la banda será desarticulada. Además, si la policía no recibe suficiente dinero, allanará la caja de caudales de los gángsters y la incautará.

El principal personaje que aparece en la pantalla es Louey, el hombre que es la mano derecha de Mugsy. Al principio del juego Louey enuncia sucintamente las reglas. El jugador toma sus decisiones estratégicas en respuesta a las solicitudes de Louey y luego espera los resultados, lapso en el que se visualiza una de las dos pantallas animadas del juego. La primera pantalla muestra una taberna clandestina en la que un gángster le dispara a un rival. En la segunda se ve una calle donde un gángster, asomándose por la ventanilla de un coche, dispara una ráfaga de ametralladora.

El mundo del hampa
Estas son dos de las escenas de Mugsy. La primera muestra parte de la fase de preguntas y respuestas del juego, en la cual Louey le está pasando a Mugsy un informe acerca de la cotización actual de los "clientes". La segunda escena pertenece a una de las secuencias animadas. Observe el contorno blanco alrededor del "hampón" de la escalera. Es un ejemplo del enmascaramiento de atributos de las celdas de caracteres.

Tras la pausa vuelve a aparecer Louey con los resultados de las decisiones del año anterior. Siempre que se hayan apartado fondos suficientes para cubrir los diversos pagos que se deben efectuar, el año acabará con beneficios. Entonces comienza la siguiente vuelta, con una repetición de la rutina de preguntas y respuestas.

El marcador del jugador se ofrece al final del programa en forma de porcentaje. El marcador depende de cuánta "pasta" y cuántos clientes se hayan acumulado, así como durante cuánto tiempo haya logrado sobrevivir Mugsy.

Aparte de las pantallas de instrucciones que poseen una pequeña ventana en la parte inferior que refleja la cantidad actual de "hampones" y "pasta en la caja", etc., las pantallas se componen enteramente de imágenes gráficas. Éstas están dibujadas en un Spectrum utilizando el paquete para gráficos *Melbourne Draw*, de Melbourne House. Dado que una pantalla en alta resolución en el Spectrum ocuparía más de seis K de memoria, es evidente que se debe de haber aplicado alguna técnica de compresión de datos para introducir el código en la memoria libre disponible.

Un examen de los gráficos de *Mugsy* revela que los programadores han recurrido a numerosas técnicas para economizar espacio. En la mayoría de los casos, las imágenes están compuestas por una serie de líneas rectas y el color está aplicado con sumo cuidado. El empleo de instrucciones de *Melbourne Draw* como FILL y DRAW permite construir imágenes con una mínima cantidad de código; DRAW, por ejemplo, sólo requiere dos coordenadas para almacenar una línea recta, mientras que el trazado de un mapa de bits exigiría el trazado de toda una serie de puntos para conseguir el mismo resultado. La forma en que el Spectrum almacena la información relativa al color determina algunos de los métodos que se utilizan: en la escena animada de la calle, surge un problema cuando un coche avanza a lo largo de la calzada mientras se ve un rostro que observa desde una ventana. Puesto que el Spectrum no permite visualizar más de dos colores en el mismo cuadrado de carácter, se ha recurrido a una "máscara" para poder cambiar los colores rápidamente. Los atributos de color (FLASH, BRIGHT, INK y PAPER) se retienen en un único byte. Para producir una máscara de fondo se debe cambiar el atributo INK, retenido como los tres bits menos significativos del byte. El byte se opera primero mediante AND con 248 para establecer en cero el color INK, y éste se vuelve a operar luego mediante AND con el nuevo color INK para producir una máscara nueva. En realidad el rostro cambia de color para igualarse al color del coche, pero esto sucede tan rápidamente que se engaña al ojo y parece que el color del rostro no cambia en absoluto.

Los gráficos de *Mugsy* son ciertamente notables, pero el juego propiamente dicho aburre en seguida. La acción es repetitiva y el jugador aprende rápidamente a manejar los diversos factores necesarios para permanecer en el juego. No obstante, *Mugsy* representa un buen ejemplo de cómo introducir gráficos de alta resolución en un espacio limitado.

Mugsy: Para el Spectrum de 48 K
Editado por: Melbourne House, Church Yard, Tring, Herts
Autores: Phillip Mitchell, Greg Cull, Clive Barrett, Russell Comte
Palanca de mando: No se necesita
Formato: Cassette

Conocer el entorno

¿Cómo combinar los "sentidos" del robot estudiados hasta ahora? Es el tema que trataremos en este capítulo

Mike Brownlow

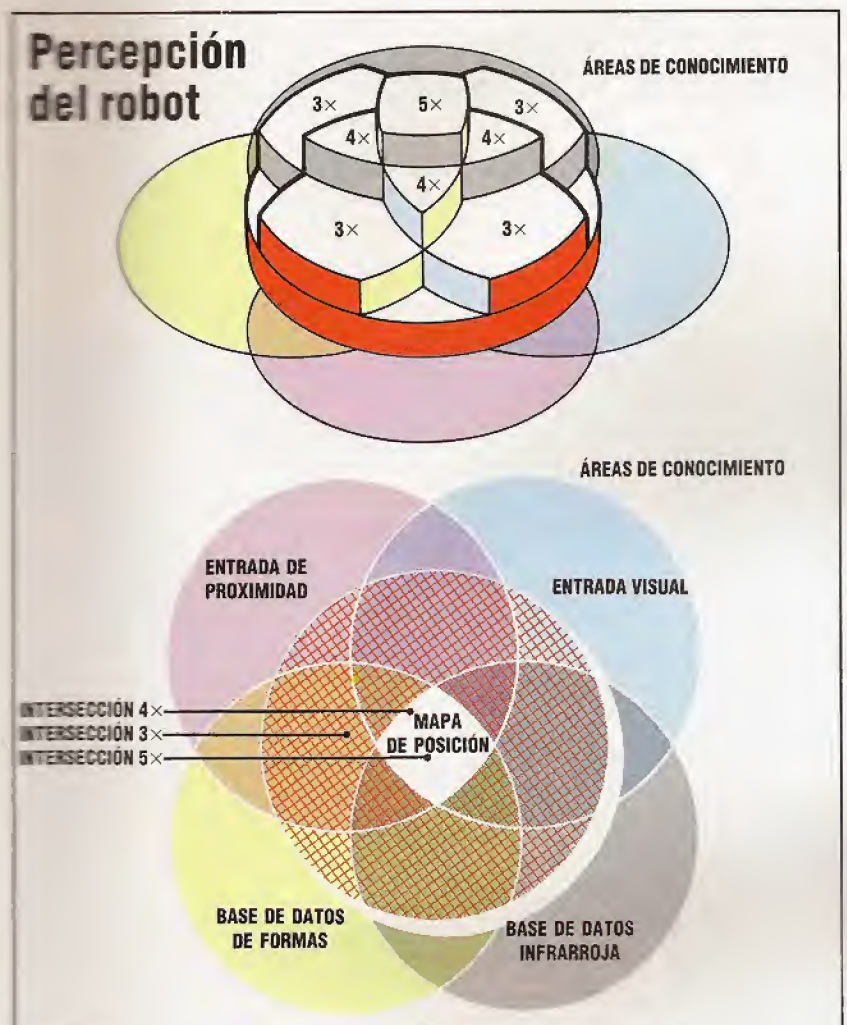
A sus órdenes...
Hemos creado un camarero-robot imaginario que debe sintetizar diversas entradas sensoriales para desempeñar diligentemente su función. Su mayor problema reside en el hecho de que los invitados están moviéndose de manera constante, lo que significa que debe actualizar su modelo interno del espacio tomando en consideración esta circunstancia

Cuando examinábamos los sensores que puede utilizar un robot para obtener cierto conocimiento del mundo en el cual se mueve, consideramos cada tipo de entrada sensorial (vista, sonido, tacto) como si se la empleara de forma aislada. Este sería un supuesto bastante factible si el robot tuviera sólo un sensor; pero, en la práctica, los más perfeccionados disponen de varios. Para comprender su entorno el robot ha de ser capaz de integrar estas entradas sensoriales empleando cada una de ellas como comprobación de las otras, con el objeto de construirse un modelo interno del mundo circundante.

Este hecho no debería sorprender, porque así es, al parecer, cómo actúan también los seres humanos. Nuestros sentidos no existen aisladamente: estamos utilizando de manera constante la entrada de un sentido como verificación de la entrada de otro, y la consecuencia es la construcción de una imagen muy completa de nuestro entorno. El más claro ejemplo de ello nos lo proporcionan los estudios realizados en personas ciegas de nacimiento a las que se ha dotado de vista gracias a una intervención quirúrgica. Estos pacientes a menudo sorprenden a sus cirujanos por la velocidad con la cual pueden hacer un uso completo de su vista. Ello se debe a que los ciegos poseen un conocimiento muy exacto del mundo que les rodea como consecuencia de poder tocar objetos, desplazarse por él y escuchar descripciones del entorno. Por consiguiente, una vez dotados de visión, pueden utilizar este conocimiento y aplicarlo para interpretar lo que perciben sus ojos.

Para poder obtener los mejores resultados con los robots, debemos posibilitar la interacción de sus sentidos como si se tratara de los de una persona. Por ejemplo, un robot diseñado para asir objetos podría ser capaz de cogerlos "a ciegas", pero será mucho mejor que disponga de un sistema visual, porque entonces podrá localizar los objetos aun cuando estén ligeramente desplazados de sitio o en un ángulo diferente a aquel en el que el robot espera hallarlos en función de su programación. Para hacer esto, el robot debe construirse alguna clase de modelo interno de su entorno utilizando las entradas de todos sus sensores. Debe ser capaz de mirar el objeto y reconocerlo, debe posicionar luego su efector final y efectuar los cálculos necesarios para levantar el objeto.

Percepción del robot



En la ilustración, con cinco áreas de conocimiento, hay una intersección de los cinco círculos, cuatro intersecciones distintas de cuatro círculos, etc. El robot coteja cualquier objeto que tenga a la vista con la intersección de cinco elementos, luego mediante cada una de las intersecciones de cuatro elementos, y así sucesivamente hasta conseguir una pareja

Kevin Jones

Interface de intersección
Normalmente los robots tienen varias fuentes de conocimiento a su disposición: algunas son sus bases de datos preprogramados (de siluetas de objetos comunes y "firmas" infrarrojas, p. ej.), otras son bases de datos empíricas (como el mapa corriente de su entorno que posee el robot) y unas terceras son canales de entradas sensoriales (como proximidad y forma de objetos). En el caso ideal (cuando el robot "conoce" su posición y "comprende" su mundo circundante), la intersección de estas áreas de conocimiento debe identificar con carácter exclusivo cualquier objeto que haya a la vista del robot.

La ilustración más simple de este modelo interno es la del robot que salva laberintos (véase p. 1252), que utiliza sensores para calcular la posición de las paredes del laberinto, construyendo un mapa interno bidimensional a medida que va avanzando. Si ampliamos esta idea a un brazo, el mapa debería ser tridimensional. Al dotar de vista al robot el mapa tridimensional adquiere inmediatamente colores, variaciones de brillo y patrones que ningún sensor táctil puede detectar. Con cierta medida de reconocimiento de habla, el robot puede agregar información hablada a su modelo del mundo circundante.

Uno de los problemas con los que se enfrentan los diseñadores cibernéticos que están tratando de posibilitar que el robot interprete su entorno es que el mundo no es estático y cambia de forma constantemente. Por consiguiente, es necesario que el robot, asimismo, esté equipado con algún medio de dar cabida a tales cambios.

Si consideramos un robot que esté programado para llevar a cabo alguna tarea sencilla, como apilar ladrillos, se hace evidente la magnitud de este problema. Si los ladrillos son de distinto tamaño se los debe colocar uno encima de otro muy cuidadosamente, y si el centro de gravedad de cada uno de ellos se desplaza fuera de la superficie de la base de los ladrillos, toda la pila se derrumbará. Pero ¿qué puede saber un robot acerca de las leyes de la gravedad? Y si la pila se viene abajo, ¿comprenderá lo que ha sucedido y emprenderá la acción necesaria?

Resolución de problemas

Para abordar este problema existen dos enfoques principales. El primero es programar el robot con datos que incluyan un curso de acción prescrito para cualquier eventualidad. Esto obviamente limitará el entendimiento del robot a ciertas tareas claramente definidas. El robot apilador de ladrillos se programará, entonces, con instrucciones para asegurar que cada ladrillo se coloque exactamente encima del que se encuentra debajo, con el centro de gravedad de uno situado directamente encima del centro de gravedad de otro.

El segundo enfoque es el que defienden quienes sostienen que la única forma de que un robot pueda llegar alguna vez a comprender su entorno es aprendiéndolo por sí mismo. Éste es un campo de la ciencia informática que se denomina aprendizaje o *heurística*. Según este enfoque, el robot se programa para llevar a cabo una tarea determinada y se le proporciona realimentación, ya sea proveniente de una persona o bien de su propio sensor, que le dice lo bien que lo ha hecho. En relación a esta realimentación el robot modificará su propio programa interno (su propio modelo del mundo) con el fin de mejorar su rendimiento y construirse una "biblioteca de referencia" que le ayudará a hacer frente a futuras tareas. El robot que salva laberintos actúa de esta forma cuando intenta hallar la mejor ruta a través de un laberinto. Utilizando sus sensores puede detectar cualquier callejón sin salida, emprendiendo la acción necesaria para desandar el camino y volver a intentarlo con uno nuevo. Lamentablemente, no existe ningún programa de aprendizaje que se pueda utilizar cada vez que un robot necesita aprender una tarea.

Después de que el robot ha "aprendido" la lección correspondiente, ha de almacenarla, sea cual fuere, en forma de programa para ordenador. Esta tarea se conoce como *representación del conocimiento*. Tradicionalmente, el conocimiento del robot se puede almacenar como líneas de código similares a las de un programa común para ordenador. Pero las técnicas de la inteligencia artificial han llevado a otros enfoques. Existe en uso una amplia gama de técnicas, pero las más comunes incluyen *reglas de producción, redes semánticas* y *marcos*.

Las reglas de producción responden a la forma de las construcciones IF...THEN y son simples sentencias de la realidad. Por consiguiente, un robot podría almacenar su conocimiento en la forma: SI (IF) hay una pared de ladrillos delante de ti, ENTONCES (THEN) no puedes avanzar. Puede haber toda una secuencia de reglas de este tipo; ofrecen la ventaja de ser fáciles de escribir y fáciles de comprender para el programador que esté escribiendo el programa. Pero existe el pequeño problema de que el robot también tiene que comprenderlas: necesita lo que suele llamarse un *motor de inferencias* para ser capaz de interpretar estas reglas como un curso de acción. Los programas que utilizan reglas de producción se pueden escribir en un lenguaje convencional, como el BASIC, pero lo más común es

que se escriban en un lenguaje *declarativo*, como el PROLOG, que está mejor diseñado para tratar esta clase de conocimiento. Ello se debe a que, a diferencia de los tradicionales, los lenguajes declarativos no ejecutan sus instrucciones de una en una. En cambio el programa busca continuamente un conjunto dado de circunstancias al cual se puedan aplicar una o más reglas. Cuando esto sucede, esa regla determinada "se dispara" y es ejecutada, lo que, a su vez, puede conducir a que se "disparen" otras reglas.

Las redes semánticas son una forma de estructuras gráficas que se utilizan para representar el conocimiento, y se puede pensar en ellas como una red de relaciones entre diversos elementos del conocimiento. La razón por la cual se denominan redes semánticas, en vez de simplemente redes o gráficos, reside en que los enlaces individuales pueden poseer algún significado en sí mismos. Un arco que enlace dos nudos puede ser un arco que indique la existencia de una clase especial de relación entre aquellos dos nudos. Por lo tanto, un nudo etiquetado "mesa" podría estar unido a un nudo etiquetado "mueble". En este ejemplo, la relación es que una mesa es un tipo de mueble; de modo que podemos decir que el enlace es un enlace de "tipo".

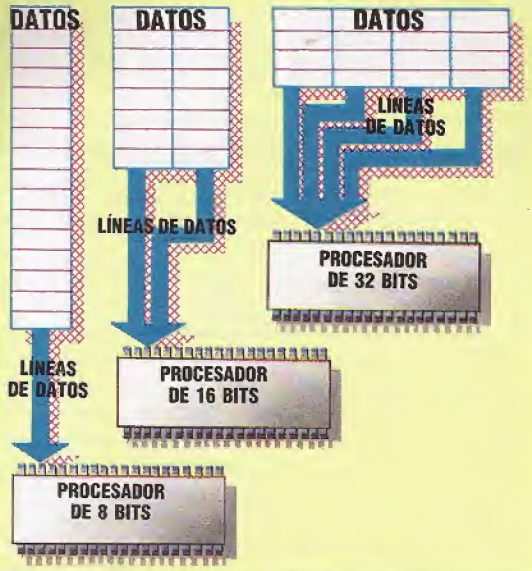
Esta clase de representación del conocimiento se puede programar en un lenguaje convencional. Usted puede intentarlo empleando el BASIC y representando los diversos nudos y enlaces mediante variables en serie. Pero lo más común es utilizar uno de los lenguajes de la inteligencia artificial, como el LISP, porque hacen que la expresión de estas complejas relaciones con nombre sea mucho más sencilla.

Los marcos son una especie de cuestionario en blanco que se diseña específicamente para cada tipo de situación con la que se pueda enfrentar un robot. La idea es muy fácil de comprender y se podría programar de inmediato en BASIC utilizando simples matrices en serie bidimensionales: una dimensión para la "pregunta" y otra para las "respuestas".

En este enfoque se considera que el robot no posee un conocimiento completo acerca de una situación hasta haber rellenado todos los puntos del cuestionario. Sólo entonces podrá emprender la acción apropiada. Un refinamiento de este método consiste en poner a disposición del robot una gran selección de marcos. Una de las tareas del mismo será, entonces, seleccionar el marco adecuado para una situación dada.

Un aspecto importante de la representación del conocimiento, del cual hemos hablado muy brevemente, es el papel que representan los lenguajes de programación. El LISP, por ejemplo, al ser un lenguaje de proceso de listas, es especialmente adecuado para esta clase de enfoque para el almacenamiento y la recuperación de datos. La elección de lenguaje, en consecuencia, simplificará la representación del conocimiento de determinadas formas.

En este capítulo hemos analizado unos cuantos métodos para programar al robot con un entendimiento más completo de su entorno. Este tema todavía sigue siendo objeto de considerable investigación en los departamentos de ciencias de universidades de todo el mundo y los elementos clave son la utilización de sensores, la realimentación, el aprendizaje y la representación del conocimiento.



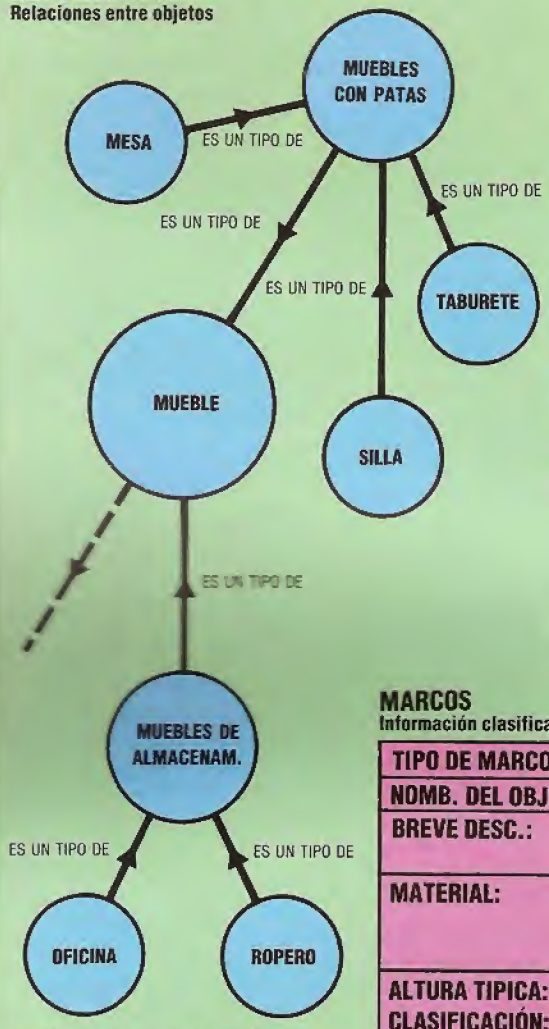
Líneas paralelas

Los datos que entran en el sistema de proceso del robot a través de sus sensores se deben procesar con la suficiente rapidez para permitir una reacción inmediata ante los datos entrantes.
La longitud de la cola de datos está determinada por la complejidad del algoritmo que los interpreta y la velocidad de proceso. El volumen de datos generados por un robot complejo puede ser tal que un procesador de ocho bits como el Z80 o el 6502 no lo puedan afrontar, formándose, de este modo, larguísimas colas de datos. La solución de este problema está en el empleo de procesadores de 16 o 32 bits

Caminos del conocimiento

El conocimiento se puede definir como la información dentro de un contexto. Los ordenadores están construidos para almacenar información pero no están especialmente dotados para relacionar los datos, convirtiendo la información en conocimiento. Por consiguiente, se deben inventar métodos apropiados para representar el conocimiento. Uno de estos procedimientos es la red semántica, que se puede almacenar como una lista encadenada; otros son los marcos, que son simplemente matrices bidimensionales; también están las reglas de producción, que son listas de información y operadores lógicos.
Ninguno de estos métodos es el ideal para representar todo el conocimiento, y las combinaciones de métodos son frecuentes. Aquí utilizamos una red semántica para representar un conocimiento detallado. Observe que estas representaciones son estáticas

RED SEMÁNTICA
Relaciones entre objetos



MARCOS
Información clasificada

TIPO DE MARCO:	1
NOMB. DEL OBJ.:	TABURETE
BREVE DESC.:	3 PATAS ASIENTO
MATERIAL:	MADERA METAL PLÁSTICO
ALTURA TIPICA:	0,5-1 m
CLASIFICACIÓN:	MUEBLES CON PATAS ASIENTOS

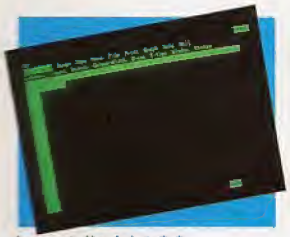
REGLAS DE PRODUCCIÓN
Listas booleanas

IF TABUR. **THEN** (3 PATAS **AND** ASTO.) **AND** (MAD. **OR** METAL **OR** PLAST.)

Gran macro

Examinemos el uso de macro de teclado y otras características en el paquete «1-2-3», producido por Lotus

Los programas de hoja electrónica que hemos examinado hasta ahora han sido diseñados para micros personales como el BBC, el Spectrum, el Commodore 64 y el Sinclair QL. Estos programas se ven necesariamente entorpecidos por la limitada cantidad de RAM disponible para el programa en sí y para los modelos desarrollados. Programas similares escritos para el IBM PC, el ACT Apricot y otras máquinas de oficina pueden sacar partido de grandes memorias residentes y una mayor velocidad de proceso. A consecuencia de ello, algunos de los programas de modelos más innovadores sólo se pueden ejecutar en costosas máquinas de esta clase. Un ejemplo es el *Lotus 1-2-3*, un programa integrado de hoja electrónica, base de datos y gráficos, que estudiamos someramente en p. 1124.



La pantalla del *1-2-3*
El menú principal se trae a la pantalla pulsando la tecla /, como en el *VisiCalc*

Economía a través de macros
Esta es nuestra macro para asignar nombres a las zonas tal como aparece en la hoja de trabajo. Si fuera necesario, las instrucciones de la macro se podrían colocar a lo largo de una columna

El *1-2-3* impone considerables demandas de memoria del ordenador por la propia naturaleza de su diseño. Además de exigir suficiente RAM para manipular el código de sus tres aplicaciones principales, necesita espacio para una hoja de trabajo con una capacidad máxima teórica de 256 columnas por 1 028 filas. La memoria se le asigna a la hoja de trabajo sólo a medida que es necesaria; pero, aun así, las primeras versiones del *1-2-3* requieren un mínimo de 128 K sólo para funcionar y las últimas versiones exigen al menos 256. Los costos de un programa como éste y un sistema para ejecutarlo son sumamente onerosos, pero el resultado es un paquete para el usuario con una amplia gama de características. En este capítulo le enseñaremos a utilizar una de las facilidades más interesantes del *1-2-3*, la macro de teclado. Pero para comprender las macros primero debemos examinar la forma en que opera este programa.

Encendido

Cuando se lo activa, el *1-2-3* visualiza el Lotus Access System, un conjunto de instrucciones para la gestión de datos. Colocando el cursor sobre la primera opción, 1-2-3, y pulsando Return, se carga la hoja de trabajo en la memoria y se prepara la visualización en pantalla. Las filas del *1-2-3* están etiquetadas con letras y las columnas están numeradas. Al igual que en el *Multiplan*, que analizamos anteriormente (véase p. 1244), el *1-2-3* es activado por menú. El menú principal se visualiza al pulsar la tecla /. A partir de entonces, las opciones del menú se seleccionan o bien digitando la primera letra de la instrucción apropiada, o colocando el cursor sobre la instrucción y pulsando Return.

El *1-2-3* posee tantas opciones de instrucciones que hay varios niveles de submenús. Esto significa que el usuario puede utilizarlo para efectuar centenares de tareas, pero también significa que algunas operaciones requieren una gran cantidad de pulsaciones de teclas (véase diagrama). Para ilustrar este punto, tomemos un ejemplo. El *1-2-3* permite designar a una celda o a una zona de celdas con una etiqueta identificadora. Cuando se desea actuar sobre la zona designada (incluyéndola en una fórmula, p. ej.) se emplea el nombre asignado en lugar de la referencia a la celda, así:

A3–B3=C3 — referencias a celdas
VENTAS–COSTO=BENEFICIO — referencias a nombres

La asignación de nombres a las zonas simplifica y acelera la tarea de buscar cosas en una gran hoja de trabajo. Para asignar un nombre a un grupo de cuatro celdas en el *1-2-3*, se requieren las siguientes pulsaciones de tecla:

/ — visualiza el menú principal;
R(ange,serie) — le dice al 1-2-3 que se va a efectuar una operación sobre un pequeño grupo de celdas, en vez de sobre la hoja de trabajo completa;
N(ombre) — a la serie de celdas identificadas se le va a asignar un nombre;
C(rear) — preparar al 1-2-3 para aceptar un nombre y asignarlo;
Digite el nombre a asignar: "COMIENZO", por ejemplo;
Return para aceptar la celda activa como el principio de la serie;
Cursor a la derecha cuatro espacios;
Return para aceptar la celda activa como el final de la serie.

Por consiguiente, este proceso exige 10 pulsaciones de teclas, más el nombre propiamente dicho.

Para reducir este proceso a un número más manejable, el *1-2-3* permite el empleo de macros de teclado. Las macros son como programas simples, escritos en el lenguaje operativo del paquete. Se crean almacenando las pulsaciones de teclas requeridas en una pequeña porción de la hoja de trabajo, asignándole un nombre a la posición, asignando luego el nombre a una tecla específica. A partir de ese momento, el *1-2-3* efectuará las pulsaciones de forma automática cada vez que pulse la tecla asignada juntamente con una de función especial, etiquetada ALT en el IBM PC y en las máquinas compatibles con el mismo.

Para automatizar el proceso de asignación de nombres a celdas, comenzamos por destinar para la macro una selección de celdas de la hoja de trabajo. Estas celdas se deben elegir cuidadosamente por dos motivos. En primer lugar, deben ocupar un espacio a salvo dentro de la hoja de trabajo, una zona en la que jamás se colocarán datos. En segundo lugar, como hemos mencionado, el *1-2-3* sólo destina espacio de memoria a las celdas que se activan. Una celda se activa siempre que sea señalada por el cursor, de modo que a los espacios vacíos entre celdas de datos se les otorgará un espacio de

la memoria. Por consiguiente, si se colocara una macro hacia la derecha, muy alejada del resto de la hoja de trabajo, las celdas vacías podrían devorar varios Kbytes de memoria utilizable. Por este motivo, a menudo es preferible colocar las macros sobre el margen izquierdo de la hoja de trabajo, en las columnas A, B y C, por ejemplo. Si todas las fórmulas de la hoja de trabajo se escriben como para funcionar de izquierda a derecha, las zonas de las macros no se tocarán en absoluto.

Vamos a construir nuestra macro de asignación de nombres en la columna A. Si la cantidad de pulsaciones se vuelve demasiado grande como para caber en una sola celda, se puede colocar a la macro en filas subsiguientes de la misma columna. Señalando el cursor a la celda A1, realizamos las pulsaciones de teclas necesarias:

```
'/ R N C
```

En este punto, el *1-2-3* aguarda para aceptar desde el teclado el nombre elegido. Se inserta una pausa entrando un signo de interrogación encerrado entre paréntesis, (?). El programa esperará hasta que el usuario pulse Return para proseguir con la macro. El formato de paréntesis se emplea coherentemente siempre que se desea una acción que no se pueda indicar mediante una tecla específica. En esto están incluidos los movimientos del cursor, que se indican digitando una palabra de dirección entre paréntesis. Los retornos de carro se indican mediante un tilde, ~. Entonces continúa la macro:

```
'/ R N C (?)~ (derecha) (derecha) (derecha) (derecha)~
```

Ahora hemos entrado todas las pulsaciones de tecla que necesitamos para asignar un nombre a una zona. Éste es el cuerpo de nuestra macro.

El siguiente paso consiste en asignarle un nombre a la zona con la etiqueta de una tecla, como N de nombre. Lamentablemente, aquí nos encontramos con un círculo vicioso. ¡Debemos realizar todas las pulsaciones de teclas que acabamos de entrar en nuestra macro designadora de series, porque aún no le hemos asignado un nombre a nuestra macro ni la hemos almacenado! Colocamos el cursor en la celda A1 y digitamos:

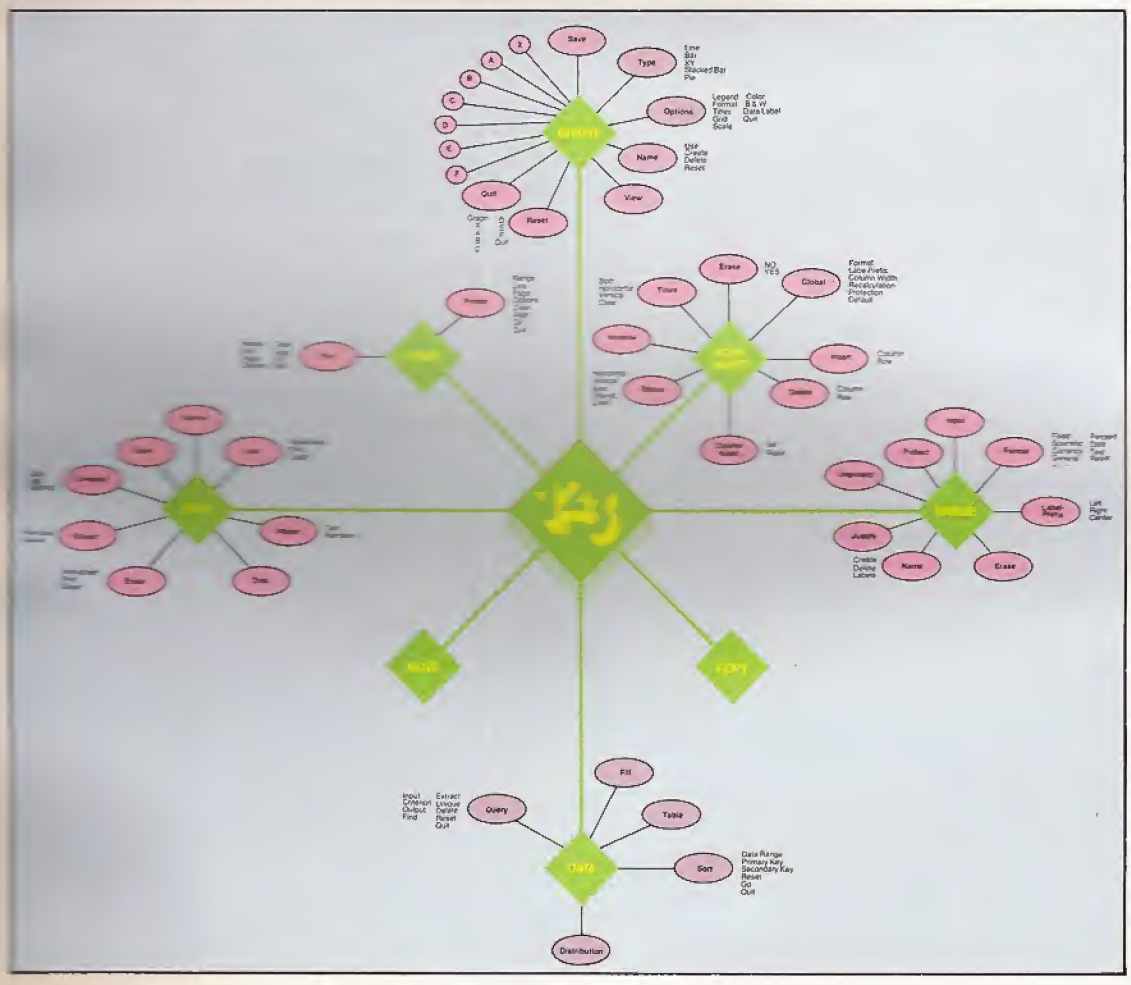
```
'/ R N C \ N Return Return
```

El carácter \ se utiliza para indicar que se debe pulsar la tecla ALT; de modo que \N, nombre de nuestra macro, significa para el *1-2-3* ALT N. Ahora que ya se le ha asignado un nombre a la zona, pulsando ALT y N juntas se activará automáticamente la secuencia de pulsaciones de teclas almacenada. A partir de este punto, la asignación de un nombre a una zona se puede efectuar digitando:

```
ALT—N NOMBRE Return
```

lo que representa una enorme mejora respecto al proceso original.

Este ejemplo, aunque útil, representa una mínima muestra del potencial de las macros de teclado. Esencialmente no existen límites en cuanto a la cantidad de pulsaciones de teclas ni a la cantidad o el tipo de operaciones que se pueden automatizar mediante macros.



Racimo de instrucciones del *1-2-3*
Tal como se refleja en la ilustración, el paquete de Lotus posee varios niveles de menús. Estas instrucciones están incorporadas. Otras las puede definir el usuario a través de las macros de teclado

Cortesía de Lotus Development Corp.

Conjurar el peligro

Nos corresponde estudiar otra fase de nuestro juego: cómo desplazarse entre escenarios y enfrentarse a los "peligros"

Ahora que ya hemos implementado la estructura básica del juego debemos considerar los procedimientos para desplazarse de un cuarto a otro cuarto. Permitiremos cuatro direcciones de movimiento: norte, sur, este y oeste.

```
TO N
  MOVE "N :LISTA.SALIDAS
END

TO S
  MOVE "S :LISTA.SALIDAS
END

TO E
  MOVE "E :LISTA.SALIDAS
END

TO O
  MOVE "O :LISTA.SALIDAS
END
```

Un procedimiento denominado MOVER verifica primero que el jugador se pueda desplazar en esa dirección, y luego deja el verdadero movimiento para otro procedimiento, MOVER1.

```
TO MOVER :DIR :LISTA
  IF EMPTY? :LISTA THEN PRINT [NO PUEDE
  AVANZAR HACIA ALLI] STOP
  MAKE "SALIDA FIRST :LISTA
  IF :DIR=FIRST :SALIDA THEN MOVER1 LAST
  :SALIDA STOP
  MOVER :DIR BUTFIRST :LISTA
END
  TO MOVER1 :NO
  MAKE :NOMBRE.CUARTO DETALLES.AQUI
  MAKE "AQUI :NO
  ASIGNAR.VARIABLES
  MIRAR
END
```

MOVER1 toma como entrada un número de cuarto. Primero vuelve a ensamblar una lista a partir de sus diversos componentes y se la vuelve a asignar al nombre del cuarto (pueden haberse producido cambios mientras el aventurero estaba en el cuarto). Luego modifica HERE para un nuevo número de cuarto y vuelve a asignar las diversas listas. El procedimiento que emplea es éste:

```
TO DETALLES.AQUI
  OUTPUT (LIST :DESCRIPCION :CONTENIDO
  :LISTA.SALIDAS)
END
```

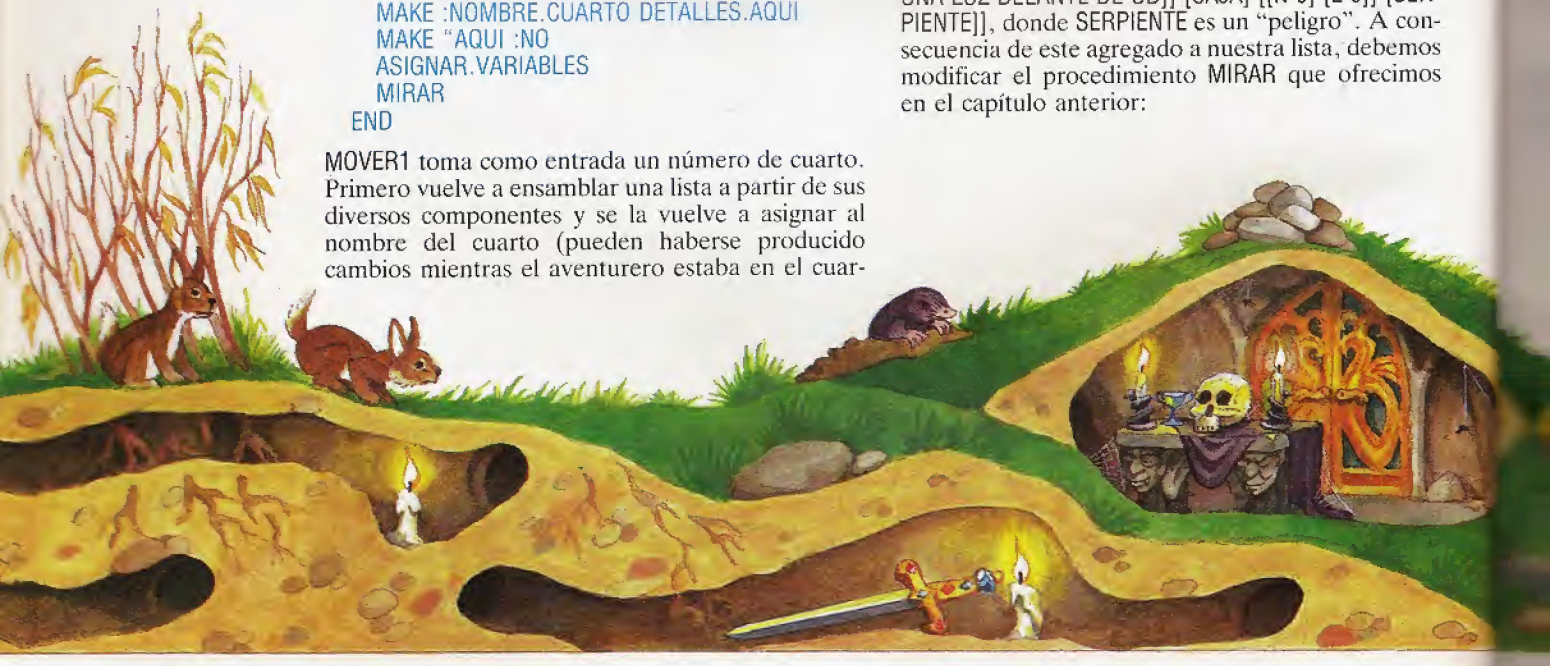
Éste utiliza la primitiva LIST, que hace una lista de sus entradas. La diferencia entre LIST y SENTENCE se explica mejor a través de un ejemplo:

```
LIST [A] [B] [C] produce [[A] [B] [C]]
SENTENCE [A] [B] [C] produce [A B C]
```

Dado que queremos conservar los componentes individuales en forma de sublistas, aquí necesitamos utilizar LIST en lugar de SENTENCE.

Los peligros del juego

Por lo general, en todo juego de aventuras hay ciertos "peligros" que evitar, como serpientes venenosas o arenas movedizas. Cuando el jugador tropieza con un peligro, es necesario activar una determinada secuencia de acciones e impedir cualquier posible movimiento para salir del cuarto hasta haber superado el peligro. La forma en que aquí hemos hecho esto es agregando otra lista a nuestra relación de cuartos, la cual contiene los nombres de todos los procedimientos de peligro especiales a ejecutar cuando se penetra en ese cuarto. Por lo tanto, debemos definir CUARTO.2 como [[[UD SE HALLA EN UNA CAVERNA OSCURA Y HUMEDA] [HAY UNA LUZ DELANTE DE UD]] [CAJA] [[N 5] [E 6]] [SERPIENTE]], donde SERPIENTE es un "peligro". A consecuencia de este agregado a nuestra lista, debemos modificar el procedimiento MIRAR que ofrecimos en el capítulo anterior:



```
TO MIRAR
  PRINTL :DESCRIPCION
  PRINT ""
  PRINT [UD VE:]
  IF EMPTY? :CONTENIDO THEN PRINT [NADA
  ESPECIAL] ELSE PRINT :CONTENIDO
  PRINT ""
  PRINT [PUEDE IR:] IMPRIMIR.SALIDAS
  :LISTA.SALIDAS
  PRINT ""
  IF PELIGRO? THEN RUN :PELIGROS
END
```

RUN es una primitiva muy poderosa del LOGO. Toma como entrada una lista y ejecuta los procedimientos que contenga esa lista. Aquí, [SERPIENTE] podría ser asociada a PELIGROS, de modo que RUN :PELIGROS ejecutaría SERPIENTE.

```
TO PELIGRO?
  IF EMPTY? :PELIGROS THEN OUTPUT "FALSE
  OUTPUT "TRUE
END
```

Ahora es necesario modificar algunos otros procedimientos para tener en cuenta estos peligros:

```
TO ASIGNAR.VARIABLES
  MAKE "NOMBRE.CUARTO WORD "CUARTO. :AQUI
  MAKE "CUARTO THING :NOMBRE.CUARTO
  MAKE "DESCRIPCION DESCRIPCION :CUARTO
  MAKE "CONTENIDO CONTENIDO :CUARTO
  MAKE "LISTA.SALIDAS LISTA.SALIDAS :CUARTO
  MAKE "PELIGROS PELIGROS :CUARTO
END

TO PELIGROS :CUARTO
  OUTPUT ITEM 4 :CUARTO
END

TO DETALLES.AQUI
  OUTPUT (LIST :DESCRIPCION :CONTENIDO
  :LISTA.SALIDAS :PELIGROS)
END

TO MOVER :DIR :LISTA
  IF PELIGRO? THEN PRINT [NO PUEDE
  AVANZAR HACIA ALLÍ: STOP
  IF EMPTY? :LISTA THEN PRINT [NO PUEDE
  AVANZAR HACIA ALLÍ: STOP
  MAKE "SALIDA FIRST :LISTA
  IF :DIR=FIRST :SALIDA THEN MOVERH LAST
  :SALIDA STOP
  MOVER :DIR BUTFIRST :LISTA
END
```

Ahora MOVER impide todo movimiento hasta que PELIGROS sea [ ]. Al preparar los peligros de esta forma podemos utilizar el mismo peligro en numerosos cuartos, y desplazarlo de cuarto en cuarto simplemente alterando las descripciones de éstos.

Ahora podemos emplear los procedimientos que hemos desarrollado aquí para construir un juego de aventuras completo llamado *El santuario de Zolloth*. En este juego el aventurero va en busca del cetro de Gilgesh, que ha sido robado por los sumos sacerdotes de Zoltoth y escondido en su templo subterráneo. El aventurero empieza el juego parado junto a la entrada de una cueva subterránea que conduce al santuario de Zoltoth. Cuando uno diseña un juego propio, puede comenzar por formular por escrito un escenario para un recorrido de éxito a través del juego, y estructurar el juego alrededor del mismo. Nosotros aquí no proporcionamos el escenario, de modo que usted está a tiempo de intentar construirlo si así lo desea.

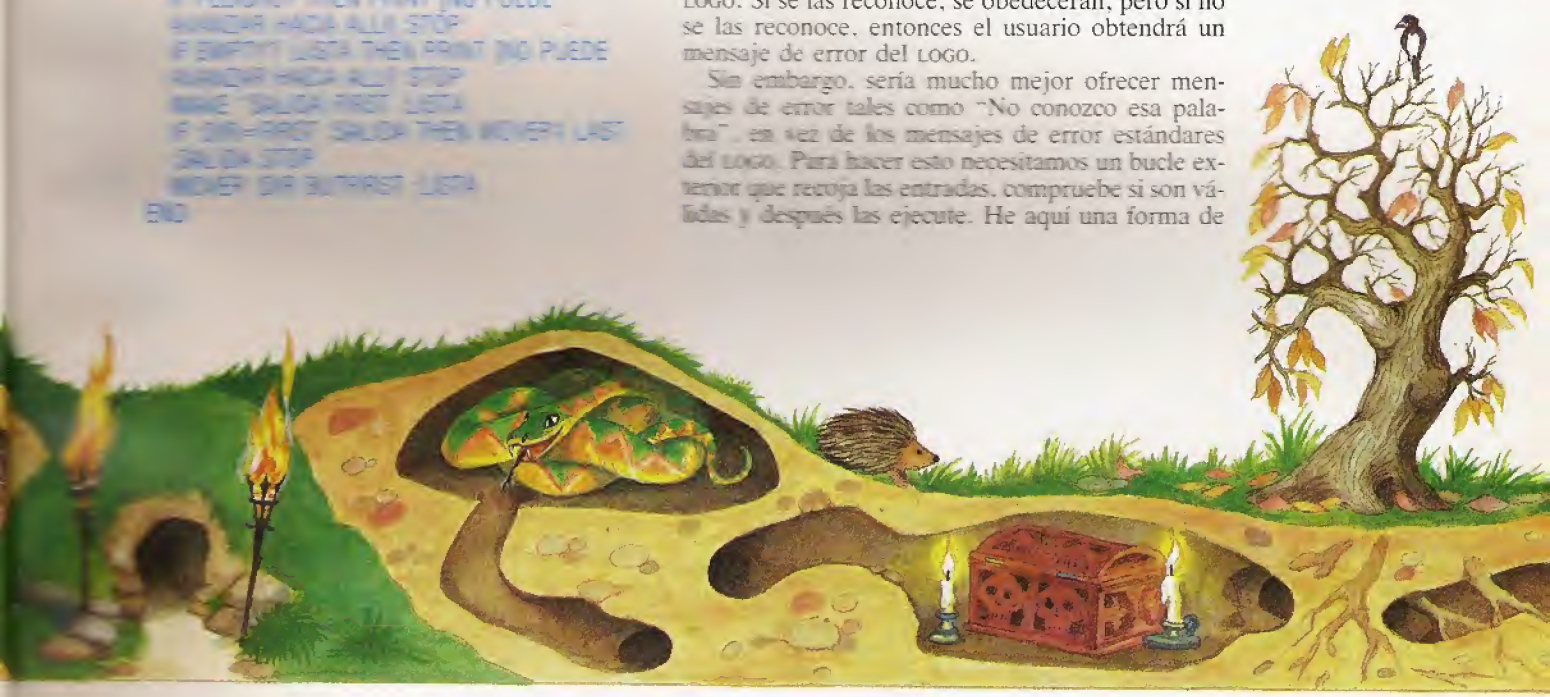
El siguiente paso consiste en planificar el juego en términos de "cuartos", es decir, escenarios dentro del juego, su contenido y posiciones en relación unos con otros. Este trazado del mundo de fantasía se utiliza luego para definir los escenarios en el programa, dando las salidas posibles desde cada escenario. Los aventureros, a su vez, habrán de trazarse un mapa a medida que vayan avanzando.

Ahora necesitamos decidir acerca del vocabulario que se empleará en el juego: ¿qué palabras expresadas por el aventurero podrá comprender el programa? Nosotros aceptaremos:

1. Siete instrucciones compuestas por una sola palabra: EMPEZAR, MIRAR, N, S, E, O, e INVENTARIO (todas ellas fueron descritas en el capítulo anterior).
2. Instrucciones de dos palabras compuestas por un verbo seguido de un sustantivo. Los verbos son: COGER, DEJAR, EXAMINAR, MATAR, FROTAR y ABRIR. Los sustantivos son: ESPADA, COFRE, CETRO, ANILLO y SERPIENTE.

Todas las instrucciones se le digitan directamente al LOGO. Si se las reconoce, se obedecerán; pero si no se las reconoce, entonces el usuario obtendrá un mensaje de error del LOGO.

Sin embargo, sería mucho mejor ofrecer mensajes de error tales como "No conozco esa palabra", en vez de los mensajes de error estándares del LOGO. Para hacer esto necesitamos un bucle exterior que recoja las entradas, compruebe si son válidas y después las ejecute. He aquí una forma de



hacer esto para el vocabulario que hemos definido hasta ahora:

```
TO EMPEZAR
  MAKE "AQUI 1
  MAKE "INVENTARIO [ ]
  ESTABLECER.CUARTOS
  ASIGNAR.VARIABLES
  MIRAR
  JUEGO
END

TO JUEGO
  PRINT1 "INSTRUCCION:
  MAKE "ENTRADA REQUEST
  IF VALIDA? :ENTRADA RUN :ENTRADA ELSE
  PRINT [NO COMPRENDO]
  JUEGO
END

TO VALIDA? :INSTR
  IF ((COUNT :INSTR)=1)THEN OUTPUT
  VAL1? :INSTR
  IF ((COUNT :INSTR)=2)THEN OUTPUT
  VAL2? :INSTR
  OUTPUT "FALSE
END

TO VAL1? :INSTR
  IF MEMBER? FIRST :INSTR [INV O E S N
  MIRAR EMPEZAR] OUTPUT "TRUE
  OUTPUT "FALSE
END

TO VAL2? :INSTR
  IF ALLOF VALV? FIRST :INSTR VALN? LAST
  :INSTR OUTPUT "TRUE
  OUTPUT "FALSE
END

TO VALN? :SUSTANTIVO
  IF MEMBER? :SUSTANTIVO [ESPADA COFRE
  CETRO ANILLO SERPIENTE] OUTPUT "TRUE
  OUTPUT "FALSE
END

TO VALV? :VERBO
  IF MEMBER? :VERBO [COGER DEJAR EXAMINAR
  MATAR FROTAR ABRIR] OUTPUT "TRUE
  OUTPUT "FALSE
END
```

El programa

Primero debe entrar todos los procedimientos que ofrecimos en el capítulo anterior (véase p. 1255). Para empezar el juego, o para reiniciarlo en cualquier momento, digite EMPEZAR.

```
TO EMPEZAR
  MAKE "AQUI 1
  MAKE "INVENTARIO [ ]
  ESTABLECER.CUARTOS
  ASIGNAR.VARIABLES
  MIRAR
END
```

ESTABLECER.CUARTOS establece los cuartos de acuerdo al mapa.

```
TO ESTABLECER.CUARTOS
  MAKE "CUARTO.1 [[[UD ESTA PARADO JUNTO A
  LA ENTRADA] [DE UNA CUEVA]] [] [[E 2]] []]
  MAKE "CUARTO.2 [[[UD SE HALLA EN UNA
  CUEVA OSCURA Y HUMEDA]] [] [[S 3] [E 4]
  [O 1]] []]
  MAKE "CUARTO.3 [[[UD SE HALLA EN UNA
  CUEVA OSCURA Y HUMEDA]] [] [[N 2] [E 5]] []]
  MAKE "CUARTO.4 [[[UD ESTA EN UNA GRAN
  CAMARA SUBTERRANEA]] [] [[N 6] [S 5] [O 2]]
  [ATAQUE.SERPIENTE]]
  MAKE "CUARTO.5 [[[UD SE HALLA EN UNA
  CUEVA OSCURA Y HUMEDA]] [ESPADA] [[N 4]
  [O 3]] []]
  MAKE "CUARTO.6 [[[UD SE ENCUENTRA EN UNA
  HABITACION DE UN SANTUARIO SAGRADO] [EN
  UN NICHO DE LA PARED NORTE] [HAY UN
  ALTAR]] [] [[N 7] [S 4] [E 8]] [PUERTA]]
  MAKE "CUARTO.7 [[[UD ESTA PARADO JUNTO]
  [AL ALTAR DE ZOLTOTH EL DORADO] [ENCIMA
  DEL ALTAR ESTA ESCRITO:] ["NO DEJEIS
  ACERCAR NINGUN METAL BASE"]] [ANILLO] [[S
  6]] []]
  MAKE "CUARTO.8 [[[UD SE HALLA EN UNA
  CUEVA OSCURA Y HUMEDA]] [] [[S 10] [E 9] [O
  6]] [ATAQUE.SERPIENTE]]
  MAKE "CUARTO.9 [[[UD SE HALLA EN UNA
  CUEVA OSCURA Y HUMEDA]] [CETRO] [[S 11] [O
  8]] []]
  MAKE "CUARTO.10 [[[UD SE HALLA EN UNA
  CUEVA OSCURA Y HUMEDA]] [] [[N 8] [E 11]] []]
  MAKE "CUARTO.11 [[[UD SE HALLA EN LA
  SACRISTIA DEL] [SACERDOTE DE ZOLTOTH EL
  DORADO]] [CETRO] [[N 9] [O 10]] []]
END
```

Complementos al LOGO

Algunas versiones de LOGO MIT no poseen EMPTY?, ITEM, COUNT ni MEMBER? En los capítulos anteriores (pp. 1234 y 1255) hemos ofrecido definiciones para las mismas. En las versiones LCSI utilice:

EMPTYP por EMPTY?
LISTP por LIST?
MEMBERP por MEMBER?
TYPE por PRINT1
AND por ALLOF
OR por ANYOF

La sintaxis IF en el LOGO LCSI queda demostrada mediante ese ejemplo:

```
IF EMPTYP :CONTENIDO [PRINT [NADA
  ESPECIAL]] [PRINT :CONTENIDO]
```

Si la condición es verdadera se ejecuta la primera lista después de la condición; si la condición es falsa, se ejecuta la segunda.
En el LOGO Atari utilice SE en lugar de SENTENCE, RL para REQUEST, y observe que ITEM no está implementada.

Sucesor del Vic-20

Demos una atenta mirada al Commodore 16, ordenador perteneciente a la "nueva generación" lanzada por su firma fabricante

Pensado posiblemente para sustituir al Vic-20, con su minúscula memoria de 3 583 bytes, el 16 comparte con el Plus/4 un BASIC muy potente, que complementa las instrucciones BASIC para tratamiento de discos existentes en máquinas como el CBM 8296 con un conjunto de instrucciones para gráficos y otras simples para sonido, haciendo al mismo tiempo un gesto simbólico hacia la programación estructurada con construcciones como DO...WHILE y LOOP...UNTIL...EXIT.

A diferencia del Plus/4, que posee un teclado y una carcasa radicalmente distintas a cualquier otra que haya producido Commodore con anterioridad, el 16 se suministra en una carcasa similar a la de los primeros modelos Vic y 64, si bien el esquema de color es diferente (color carbón, con teclas gris claro) y se ha modificado la disposición de las teclas. Las teclas son grandes, están bien situadas y poseen un tacto firme y profesional, como el de las máquinas que precedieron al 16. Una interesante innovación es la inclusión de una tecla HELP (en realidad la tecla de función F8), que ayuda al usuario, cuando el programa se interrumpe, con un informe de mensaje de error, visualizando la línea que contiene el error e iluminando de forma intermitente la parte incorrecta. En las líneas de sentencias múltiples, los caracteres se iluminan intermitentemente desde el error hasta el final de la línea; sería mucho más útil que la iluminación se restringiera al error propiamente dicho (p. ej., PRONT por PRINT).

Puntos en contra

El aspecto más controvertido tanto del 16 como del Plus/4 es el hecho de que por primera vez Commodore ha producido hardware que no es "compatible hacia arriba" con el producido previamente: ningún software del Vic ni del Commodore 64 funcionará en ninguna de las nuevas máquinas. Los conectores para palanca de mando y cassette son, asimismo, diferentes, con el primero apartándose del tipo de conector D de nueve patillas que por lo general se considera estándar. Los conectores para interface de disco y para pantalla son, no obstante, idénticos a los del 64.

De los cambios negativos y en cuanto atañe al programador de juegos, el más grave es que no existen sprites. Aunque el Vic-20 tampoco los posee, su utilización en el Commodore 64 (y en máquinas de la competencia) ha acostumbrado tanto a los usuarios a su empleo para la manipulación sencilla de formas gráficas, que ésta parece una curiosa omisión.

Al conectarse el equipo, la pantalla ofrece la ya familiar visualización Commodore, con la diferencia de que el BASIC indicado es la versión 3.5, y que hay 12 277 bytes de memoria disponibles. El BASIC 3.5 es en realidad la quinta versión que se escribe para máquinas Commodore. La versión original, 1.0, no contenía instrucciones para tratamiento de discos, y éstas eran muy torpes en la versión 2, que por algún motivo fue la versión que se incorporó en el Vic-20 y el Commodore 64. Hacia marzo de 1980, la versión 2 había sido superada por la versión 4, que es un BASIC muy eficaz escrito para los ordenadores de la serie Pet de 80 columnas, el 8032, 8096 y ahora el 8296.

La versión 3.5 es casi idéntica al BASIC 4, con unas cuantas instrucciones extras. En conjunto, la nueva versión posee unas 50 instrucciones y funciones más que las que ofrecía el Vic, incluyendo instrucciones de ayuda utilizadas para escribir y depurar programas, características de programación estructurada, instrucciones para gráficos y sonido.

La instrucción directa MONITOR invoca a Tedmon, el monitor residente (al cual también se puede acceder mediante SYS 4, como en la serie Pet). Éste emplea instrucciones mnemónicas de una sola letra: por ejemplo C, que compara dos secciones de la memoria e informa sobre las diferencias, y S, que guarda en cinta o en disco.

Las rutinas *kernal* básicamente no han sido modificadas. Estas rutinas, que se llaman desde código máquina, gobiernan el tratamiento de las rutinas de entrada y salida. No obstante, se ha agregado en $FF81 la función IOINIT del 64, para inicializar los dispositivos de entrada/salida (y especialmente los cartuchos de programas).

Commodore 16
Destinada a sustituir al Vic-20 en la línea de Commodore, la nueva máquina posee 16 K de memoria y un BASIC perfeccionado. El Commodore 16 es compatible con el Plus/4 tanto en software como en enchufes, pero no así con las máquinas Commodore más antiguas, el Vic-20 y el Commodore 64

Chris Stevens

Gráficos

La pantalla de alta resolución, trazada por mapa de bits, tiene unas dimensiones de 320 por 160 pixels, y la pantalla de colores múltiples da una resolución de 160 por 160. La instrucción de modo GRAPHIC es obviamente más fácil de invocar que las PEEK y POKE del 64, así como la pantalla dividida, si bien con ésta el texto queda limitado a las cinco líneas inferiores. No obstante, en una pantalla para gráficos se puede colocar texto en cualquier sitio mediante el empleo de la sentencia CHAR, de modo que

```
CHAR 1,0,0, "ESTA ES LA LINEA SUPERIOR"
```

se imprimirá a lo largo de la parte superior de la pantalla, siendo 1 el color seleccionado, y aludiendo los dos ceros a las posiciones de columna y fila. La serie se puede imprimir en video invertido si se la señala con un ',1'; esto se desactiva mediante ',0'. Cualquier error de sintaxis en alguna de las modalidades para gráficos devolverá al usuario a GRAPHIC 0, la pantalla de "texto puro".

La instrucción DRAW es una especie de compromiso entre las líneas rectas, bastante limitadas, disponibles en el Amstrad, y la DRAW MSX similar al LOGO. Por ejemplo, se puede dibujar un cuadrado mediante:

```
DRAW,10,10 TO 10,60 TO 60,60 TO 60,10 TO 10,10
```

En este caso no está definido el primer parámetro, de modo que el color del cuadrado será el último color establecido. Este color se puede modificar insertando el valor que interese. Existe asimismo una instrucción BOX (caja), que se utiliza específicamente para dibujar rectángulos especificando las posiciones de los cuatro vértices, con un parámetro de "relleno" para pintar la caja con un color.

La instrucción CIRCLE dibuja elipses, octágonos e incluso rombos y triángulos además de círculos propiamente dichos, a tenor de los parámetros que se especifiquen. Las formas no circulares se eligen especificando ángulos de 120° entre segmentos para un triángulo, 90° para un rombo y 45° para un octágono. El valor por defecto es 2°. PAINT rellenará la forma así creada, ya sea con el mismo color que la forma esbozada o bien con un color de primer plano definible, y las formas se pueden guardar (SAVE) o recuperar desde disco mediante el empleo de las instrucciones SSHAPE y GSHAPE.

Los colores se especifican desde BASIC asignándoles uno de entre 16 valores al fondo, primer plano, multicolor 1, multicolor 2 o borde, con un parámetro de brillo opcional de 0 a 7. El brillo por defecto es 7 (el más intenso). En todas las instrucciones para dibujar, el parámetro de color se debe escoger entre una de las cinco áreas ya definidas.

Sonido

Después del nivel de sofisticación de las instrucciones para sonido del chip SID (Sound Interface Device) del 64, el sonido de los canales del Commodore 16 es más bien una decepción, especialmente dado que la actual generación de máquinas competitivas, que utilizan el chip de sonido de General Instruments (Amstrad, MSX, Einstein), ofrece tres canales más ruido.

Sin embargo, la instrucción SOUND no exige la colocación (POKE) de cifras en las posiciones 54272

Bien venido el cambio

Al cabo de numerosas quejas formuladas por programadores en relación a las deficiencias de sus ROM de BASIC, finalmente Commodore ha editado una versión comparable a aquellas que ya son habituales en otros micros personales. Todas las adiciones son bien venidas. Inexplicablemente, los procedimientos definidos por el usuario se han omitido en esta versión de BASIC, que, con esta salvedad, ya es aceptable. Las nuevas instrucciones, incluidas las del BASIC 4, son las siguientes:

Funciones	Ayuda	Estructura	Gráficos	Del BASIC 4
RGR	AUTO	DO	GRAPHIC	DIRECTORY
RGCL	TRON	WHILE	SCNCLR	DSAVE
RLUM	TROFF	LOOP	PAINT	DLOAD
JOY	HELP	UNTIL	CHAR	HEADER
RDOT	MONITOR	EXT	BOX	SCRATCH
INSTR	DELETE	ELSE	CIRCLE	COLLECT
DEC	RENUMBER	PUDEF	GSHAPE	COPY
HEX$	KEY	USING	SSHAPE	RENAME
SOUND	ERR$		DRAW	BACKUP
VOL	TRAP		LOCATE	
	RESUME		COLOR	

Chris Stevens

COMMODORE 16

DIMENSIONES
76,2 x 203,2× 406,4 mm

CPU
MOS 7501, 0,89 a 1,76 MHz

MEMORIA
16 K de RAM (12 K de memoria para usuario), 32 K de ROM

PANTALLA
Texto: 25 filas de 40 columnas. Gráficos: 320 por 160 pixels. Cinco modalidades: texto, alta resolución, alta resolución con cinco líneas de texto, multicolor, multicolor con cinco líneas de texto, 15 colores x 8 niveles de brillo, más negro = 121 tonal.

INTERFACES
Puerta en serie Commodore, ranura para ampliación de cartucho de ROM/memoria, puerta para interface de unidad de cassette (8 patillas), 2 puertas para palanca de mando (8 patillas), salida para pantalla: compuesto/crominancia/brillo/audio, salida RF con interruptor para regulación alto/bajo, entrada alimentación eléctrica (9 V)

LENGUAJES DISPONIBLES
Intérprete de BASIC 3.5 en ROM, 75 instrucciones, incluyendo trazado completo de gráficos

TECLADO
Tipo máquina de escribir, 66 teclas, incluyendo 7 teclas de función reprogramables y HELP

VENTAJAS
BASIC avanzado, instrucciones excelentes para tratamiento de discos, instrucciones simples para sonido y para gráficos, fácil acceso a la pantalla para programación en lenguaje máq.

DESVENTAJAS
No es compatible con los equipos previos de CBM, razón por la cual dispone de muy poco software. Conectores de E/S incompatibles; no posee sprites

a 54296, al igual que en el 64. Si se conoce la frecuencia de una nota, puede buscarse en una tabla del manual y utilizar la cifra proporcionada para definir la nota a reproducir. Por ejemplo:

```
SOUND 1,770,60
```

tocará la nota *la* (a una frecuencia de 440 Hz) durante 60 sesentavos de segundo (es decir, un segundo) en el canal uno.

El sonido más bajo que se puede ejecutar es *la* dos octavas por debajo de *do central* (110 Hz), y el más alto es *sol* dos octavas arriba de *do central* (1 575 Hz), dando una envergadura musical total de cuatro octavas. Hay disponibles dos canales de música (1 y 2) o un canal de música (1 o 2) y un canal de ruido blanco (3). Ambos canales están combinados, dado que la señal de salida de audio es mono, y no existe ninguna manera de separarlos.

El Commodore 16 es una máquina atractiva, con un BASIC muy avanzado y buenas instrucciones para gráficos, pero sus facilidades para sonido son bastante primitivas, aun comparándolas con las del Vic-20, si bien en la nueva máquina son más fáciles de ejecutar.

En el momento de su lanzamiento, era muy poco el software existente para el Commodore 16. A los poseedores de Vic 20 que deseen superar y poner al día sus expectativas en informática adquiriendo esta máquina, se les debe advertir que en la misma no se ejecutarán sus antiguos programas.

La línea de la trama

Llegados a este punto, desarrollaremos una utilidad que nos permitirá formatear la salida a la pantalla de los juegos creados

Dado que tanto *Digitaya* como *El bosque encantado* son aventuras basadas en texto, emplean palabras para describir los escenarios y los acontecimientos. Pasar esta información a la pantalla utilizando sentencias PRINT sería poco elegante. Por ejemplo, una sentencia PRINT cuya longitud supere a la de una línea de la pantalla continúa en la línea siguiente, con frecuencia dividiendo en dos aquellas palabras que quedan al final de la línea de pantalla. Una laboriosa forma de abordar este problema sería considerar cada sentencia PRINT del programa de forma individual y formatear "manualmente" la salida de modo que no se cortaran las palabras del final de cada línea. Ello no representaría mayor problema si sólo hubiera de hacerse en unas pocas ocasiones, pero en un programa para un juego de aventuras ello será necesario en innumerables ocasiones. La alternativa es diseñar una rutina que nos formatee la salida. Para emplear una rutina de este tipo debemos ser capaces de pasar la frase que deseamos formatear a la rutina a través de una variable en serie, y la rutina deberá encargarse del formateado y de la salida.

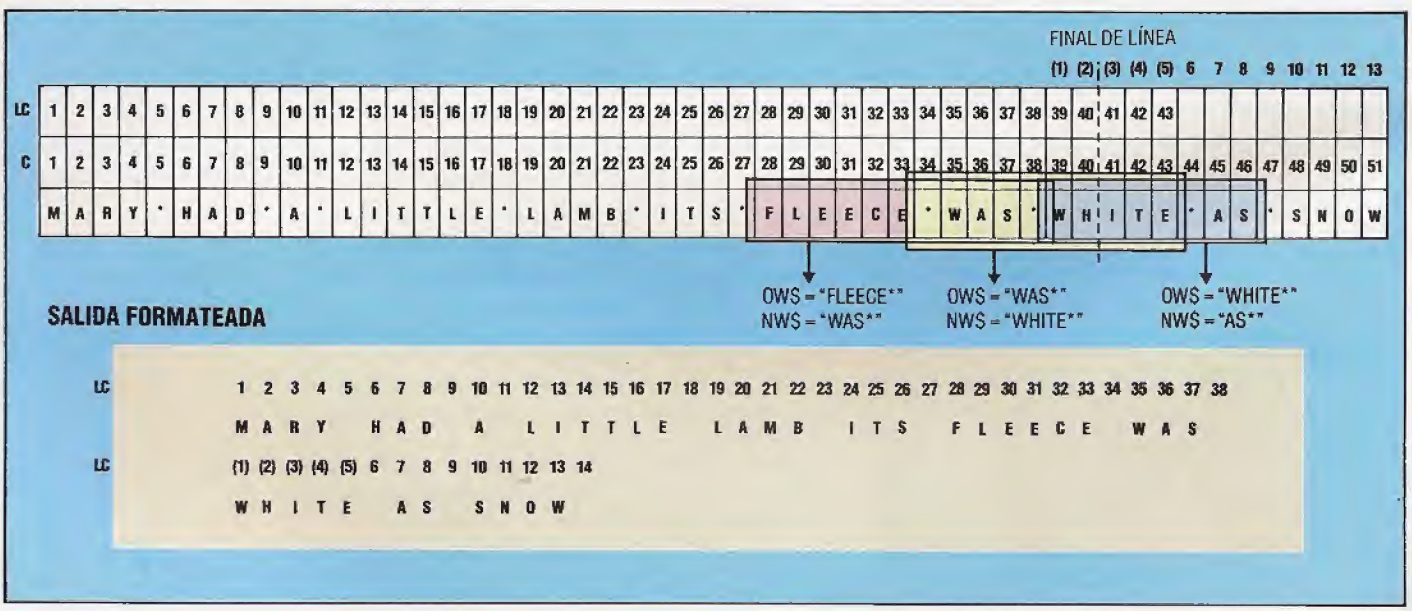
Tanto *Digitaya* como *El bosque encantado* utilizan una rutina especial para formatear su salida, de modo que antes de seguir describiendo la programación del juego será mejor que analicemos cómo trabaja esta rutina. Éste es el listado para *El bosque...*

```
5500 REM **** S/R FORMATEADO DE SALIDA ****
5510 LC=0:          REM CONTADOR CAR/LINEA
5520 OC=1:          REM VALOR INICIAL CONTADOR ANTIGUO
5530 OW$="":        REM VALOR INICIAL PALABRA ANTIGUA
5540 LL=40:         REM LONGITUD LINEA
5550 SN$=SN$+ " FICTICIA "
5560 PRINT
5570 FOR C=1 TO LEN(SN$)
5580 LC=LC+1
5590 IF MID$(SN$,C,1)=" " THEN GOSUB5800
5600 NEXT C
5605 PRINT
5610 RETURN
5620 :
5800 REM ** S/R CONTROL FINAL DE LINEA **
5810 NW$ = MID$(SN$,OC, C-OC+1):REM NUEVA PALABRA
5820 IF LC<LL THEN PRINTOW$;:GOTO5840
5830 PRINTOW$:LC=LEN(NW$)
5840 OC=C+1:OW$=NW$
5850 RETURN
```

En primer lugar, la rutina busca a través de la frase, que le ha sido pasada por la variable SN$, un carácter espacio. Cada vez que encuentra un espacio se llama a la subrutina de la línea 6020. Esta subrutina lleva a cabo varias tareas importantes. Utilizando OC para indicar el comienzo de una palabra (inicialmente, OC está establecida en 1), y C para llevar el registro del carácter que está siendo objeto del examen, se puede aislar la palabra que se ha encontrado antes del espacio utilizando MID$ y almacenarla en NW$ (por *New Word*: palabra nueva). Antes de visualizar en la pantalla el contenido de NW$, el mismo se transferirá a OW$.

Se emplea un contador de línea, LC, para contar cuántos caracteres se han utilizado hasta el momento en cualquier línea dada, cantidad que se verifica en la línea 6040 para asegurar que sea inferior a la longitud de línea permitida, LL. De ser éste el caso, entonces se imprime OW$, seguida de un punto y coma para asegurar que cualquier salida que venga a continuación continuará en la misma línea. Si LC sí fuera mayor que LL, nuevamente se imprimiría OW$, pero en esta ocasión omitiendo el punto y coma (y, por tanto, cualquier salida que venga a continuación empezará en una línea nueva). Además, LC, el contador de línea, se vuelve a establecer en la longitud de la palabra nueva.

Visualización en formación
La rutina para formatear la pantalla que utilizan *Digitaya* y *El bosque encantado* permite formatear cualquier salida en pantalla de modo que no se produzcan cortes de palabras. Mediante el empleo de las variables OW$ y NW$, la rutina va "buscando" con una palabra de antelación respecto a la que se está visualizando. Si la siguiente palabra va a superar la longitud de línea establecida, se omite el punto y coma que suprime un retorno de carro, haciendo que se inicie una línea nueva



Liz Dixon

Veamos ahora cómo funciona esta subrutina en la práctica. La rutina explora la frase a formatear, en busca de un espacio. Cuando encuentra uno, se vuelve a designar los caracteres entre el espacio y la última palabra hallada como si formaran una nueva palabra. La rutina, en efecto, busca palabras con una palabra de adelanto respecto a la que se está visualizando. La rutina comprueba si la longitud máxima de palabra se ha superado cuando agrega a la línea de pantalla el nuevo término. Si así fuera, la rutina hace que se inicie una línea nueva. Por consiguiente, se evita la separación de las palabras del final de cada línea. La adición de "FICTICIA" al final de la frase es importante, ya que representa una última palabra a almacenar en NW$. Los espacios antes y después de " FICTICIA " son significativos: el primero la convierte en una palabra separada y el último proporciona un espacio final a detectar por la rutina.

Tomemos a modo de ejemplo la oración *"Mary had a little lamb; its fleece was white as snow"* (Mary tenía un corderito; su vellón era blanco como la nieve). La anchura de pantalla que utilizaremos es de 40 caracteres. Si no se formateara la oración, la palabra *white* se separaría en dos, con las letras *ite* empezando una línea nueva. La rutina de formateado, sin embargo, toma dos palabras de la oración cada vez. Si consideramos las dos que preceden a *white*, entonces *fleece* se almacenará en OW$ y *was* en NW$. Habiendo comprobado que el contador, LC, no supera la cantidad de 40, se imprime OW$, seguida de un punto y coma; *was* se transfiere entonces de NW$ a OW$ y la rutina continúa explorando la frase, y se encuentra con el término *white*. En este punto, el contador LC excede de 40, lo que indica que *white* cae en un final de línea. Dada esta situación, OW$ (que ahora contiene la palabra *was*) sí se visualiza igualmente pero sin el punto y coma. Además, se reestablece el contador LC en la cantidad de caracteres de este término. La palabra *white* se transfiere a OW$, para la subsiguiente impresión de una nueva línea.

Comprobación de la rutina

Para comprobar la rutina, la emplearemos para formatear y visualizar la descripción inicial de la historia. Podemos ensamblar una frase de hasta 48 caracteres, utilizando la variable SN$, y llamar a la subrutina de formateado. Digite estas líneas:

```
1000 REM **** S/R HISTORIA HASTA AHORA ****
1010 SN$="BIENVENIDO AL BOSQUE ENCANTADO"
1020 GOSUB5500:REM FORMATEAR
1030 PRINT
1040 SN$="AL DESPERTARTE DE UN PROFUNDO SUEÑO, EL "
1050 SN$=SN$+"SUELO DEL BOSQUE ESTA SUAVE Y SECO. "
1060 SN$="TU NO SABES COMO HAS LLEGADO HASTA AQUI "
1070 SN$=SN$+"PERO SABES QUE DEBES LLEGAR AL "
1080 SN$=SN$+"POBLADO QUE HAY A LAS AFUERAS DEL BOSQUE
     PARA "
1090 SN$=SN$+"ESTAR A SALVO."
1100 GOSUB5500:REM FORMATEAR
1110 PRINT
1120 SN$="ECHAS UNA MIRADA A TU ALREDEDOR PARA ORIENTARTE "
1130 GOSUB5500:REM FORMATEAR
1140 PRINT:PRINT"PULSA CUALQUIER TECLA PARA COMENZAR"
1150 GET A$:IF A$="" THEN 1150
1160 PRINTCHR$(147):REM LIMPIAR PANTALLA
1170 RETURN
```

Para llamar a la subrutina "Historia hasta ahora" hemos de emplear estas líneas:

```
205 GOSUB 1000: REM HISTORIA HASTA AHORA
990 END
```

Listados para "Digitaya"

```
1110 GOSUB1250:REM HISTORIA HASTA AHORA
1270 END

1290 REM **** HISTORIA HASTA AHORA ****
1300 SN$="BIEN VENIDO A 'DIGITAYA'
1310 GOSUB5880:REM FORMATEAR
1320 PRINT
1330 SN$="MIENTRAS LA MAQUINA SUSURRA QUEDAMENTE,
     ECHAS UNA MIRADA EN DERREDOR."
1340 SN$=SN$+" HACIA EL NORTE Y HACIA EL SUR SE EXTIENDE
     UNA ANCHA AUTOPISTA."
1350 SN$=SN$+" TU MISION CONSISTE EN ENCONTRAR AL
     MISTERIOSO DIGITAYA"
1360 SN$=SN$+" Y SACARLO A TRAVES DE UNA DE LAS
     PUERTAS DE SALIDA PARA PONERLO A SALVO."
1370 SN$=SN$+".. PERO POR CUAL DE LAS PUERTAS ?"
1380 GOSUB5880
1390 PRINT:PRINT"PULSA UNA TECLA PARA COMENZAR"
1400 GET A$:IF A$=""THEN1400
1410 PRINTCHR$(147):REM LIMPIAR PANTALLA
1420 RETURN

5880 REM **** S/R IMPRESION FORMATEADA ****
5890 LC=0:          REM CONTADOR CAR/LINEA
5900 OC=1:          REM CONTADOR ANTIGUO
5910 OW$="":        REM PALABRA ANTIGUA
5920 LL=40:REM LONGITUD LINEA ANTIGUA
5930 SN$=SN$+" FICTICIA "
5940 PRINT
5950 FOR C=1 TO LEN(SN$)
5960 LC=LC+1
5970 IF MID$(SN$,C,1)=" " THENGOSUB6020
5980 NEXTC
5990 PRINT
6000 RETURN
6010 :
6020 REM **** S/R COMPROBACION FINAL DE LINEA ****
6030 NW$=MID$(SN$,OC,C-OC+1)
6040 IF LC<LL THENPRINTOW$;:GOTO6060
6050 PRINTOW$:LC=LEN(NW$)
6060 OC=C+1:OW$=NW$
6070 RETURN
```

Complementos al BASIC

Spectrum:
Para el listado de *Digitaya*, introduzca estas modificaciones en la Rutina de Formateo:

Reemplace SN$ por S$, OW$ por O$, NW$ por N$

```
5920 LET LL=32:REM LONGITUD LINEA
PANTALLA
5970 IF S$(C TO C)=" " THEN GOSUB 6020
6030 LET N$=S$(OC TO C)
```

En la subrutina Historia... sustituya SN$ por S$

```
1400 IF INKEY$="" THEN 1400
1410 CLS
```

Para el listado de *El bosque encantado*, reemplace los nombres de las mismas variables en serie y modifique estas líneas:

```
5540 LET LL=32:REM LONGITUD LINEA
     PANTALLA
5590 IF S$(C TO C)=" " THEN GOSUB 5800
5810 LET N$=S$(OC TO C)
```

y

```
1150 IF INKEY$="" THEN 1150
1160 CLS
```

BBC Micro:
Para la subrutina Historia Hasta Ahora, se deben introducir en *Digitaya* estos cambios:

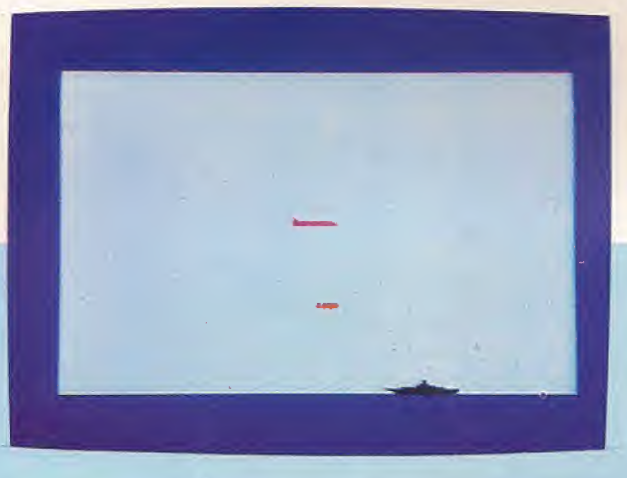
```
1095 MODE 1
1400 A$=GET$
1410 CLS
```

y en *El bosque encantado*:

```
1160 CLS
```

Exocet

El mortífero misil Exocet, que alcanzó notoriedad durante la guerra de las Malvinas, es el protagonista de este juego escrito para el Commodore Vic 20



Un portaaviones enemigo se ha aventurado por las aguas territoriales de su país y hace caso omiso a sus requerimientos. A los mandos de su Mirage 2000, debe destruirlo antes de que constituya una amenaza para su base. Para disparar pulse una tecla cualquiera.

```
 10 REM **********
 20 REM * EXOCET *
 30 REM **********
 35 GOSUB 2000
 40 GOSUB 1000
100 PRINT TAB(A)A$
105 IF B>17 THEN PRINT TAB(18)N2$;CHR$(19);:BB
    =0:GOTO 120
110 PRINT TAB(B)B$;
120 A=A-1
130 IF A<0 THEN PRINT D$+N2$+CHR$(19);:A=19
140 BB=BB+0.2
150 B=INT(BB)
160 GET X$
170 IF X$<>" " AND EX=0 THEN EX=A+7923:NX=NX-1
180 IF EX<>0 THEN 300
190 FOR I=1 TO 10
200 NEXT I
210 GOTO 100
300 EX=EX+21
310 IF EX>8184 THEN 400
320 C=PEEK(EX)
330 IF C<>32 THEN GOSUB 700
340 POKE EX-21,CN
350 POKE EX,CX
360 POKE EX+M,XC
370 GOTO 100
400 POKE EX-21,CN
410 EX=0
420 IF NX=0 THEN 500
430 GOTO 100
500 PRINT CHR$(147)
505 IF S>R THEN R=S
510 GET X$
515 IF X$<>" " THEN 510
520 PRINT:PRINT:PRINT
525 PRINT CHR$(31)
530 POKE 36869,240
535 PRINT TAB(5)"PUNTOS :"S
540 PRINT:PRINT:PRINT
550 PRINT TAB(5)"RECORD :"R
560 PRINT:PRINT:PRINT
570 PRINT TAB(5)"OTRA ?"
580 FOR I=1 TO 300:GET X$:NEXT I
600 GET X$
610 IF X$=" " THEN 600
620 IF X$<>"N" THEN :POKE 36869,254:GOTO40
630 PRINT CHR$(147);
640 POKE 36879,27
650 END
700 POKE EX-21,CN
705 S=S+10
710 POKE EX,6
715 POKE EX+M,2
720 FOR I=1 TO 30
725 X=INT(RND(TI)*4)
730 Y=INT(RND(TI)*6)
740 XY=7680+(22-Y)*22+B+X
750 POKE XY,6
760 POKE XY+M,2
770 NEXT
780 FOR I=1 TO 200:NEXT
800 NX=NX+1:PRINT CHR$(147);:GOTO 100
1000 PRINT CHR$(147);
1010 M=30720
1020 B$=CHR$(144)+CHR$(32)+CHR$(64)+CHR$(65)+CHR$(66)
     +CHR$(19)
1030 A=19:S=0:BB=0:B=0:D$=""
1060 FOR I=1 TO 10
1070 D$=D$+CHR$(17)
1080 NEXT
1085 D1$=D$+CHR$(17)
1090 A$=CHR$(156)+D$+CHR$(67)+CHR$(68)+CHR$(32)+
     CHR$(19)+D1$+D$
1100 N2$=CHR$(32)+CHR$(32)+CHR$(32)
1120 EX=0:CX=5:XC=2:NX=20:CN=32:RETURN
2000 PRINT CHR$(147);
2010 POKE 36869,254
2020 POKE 52,24:POKE 56,24
2040 FOR I=0 TO 55
2050 READ A
2060 POKE 6144+I,A
2070 NEXT
2080 FOR I=0 TO 7
2090 POKE 6400+I,0
2100 NEXT
2110 POKE 36879,30:RETURN
3000 DATA 0,0,0,0,7,255,255,127
3010 DATA 16,16,56,252,255,255,255,255
3020 DATA 0,0,0,0,224,255,252,248
3030 DATA 0,0,0,0,0,63,127,255
3040 DATA 0,0,0,1,3,255,255,255
3050 DATA 0,0,0,0,125,255,125,0
3060 8,33,128,10,0,40,0,16
```

Rupturas

Seguimos con nuestra tarea de diseño y escritura de un programa depurador de errores

Nos quedan todavía por definir dos subrutinas del módulo Puntos De Ruptura, una para eliminar puntos de ruptura previamente insertados y otra para restaurar el opcode original donde hayamos colocado un opcode SWI temporal. Debemos analizar en primer lugar la rutina que llamaremos Desinsertar Puntos de Ruptura (de una tabla de dichos puntos).

Se ha posibilitado la colocación de hasta 16 puntos en Tabla-Puntos-Ruptura (BPTAB). Para eliminar uno de ellos debemos obtener su número, que servirá de desplazamiento (dentro del intervalo del 0 al 15) en la tabla. La entrada de la tabla se elimina desplazando todas las entradas subsiguientes un lugar hacia atrás en la tabla (dos bytes) y decrementando Número-Punto-Ruptura.

Desinsertar-Puntos-De-Ruptura

Data:
Número-Puntos-Ruptura es un valor de 8 bits
Número-Punto-Ruptura es un contador de 8 bits
Tabla-Puntos-Ruptura es una tabla de direcciones de 16 bits
Entrada-a-Descartar es un desplazamiento de 8 bits (con valores entre 1 y 16)

Proceso: Desinsertar-Puntos-De-Ruptura
Decrementar Número-Puntos-Ruptura
If (si) Entrada-a-Descartar<=Números-Puntos-Ruptura (penúltimo) THEN
For (desde) Número-Punto-Ruptura=Entrada-a-Descartar To (hasta) Número-Puntos-Ruptura (penúltimo)
Llevar Tabla-Puntos-Ruptura (Número-Punto-Ruptura+1)
a Tabla-Puntos-Ruptura (Número-Punto-Ruptura)
Llevar Valores-Sustituidos (Número-Punto-Ruptura+1)
a Valores-Sustituidos (Número-Punto-Ruptura)
Endfor
Endif

Fin de proceso

El parámetro Entrada-a-Descartar puede ser pasado en B. El contador Número-Punto-Ruptura puede también colocarse en B, y adquirirá automáticamente su correcto valor inicial. Después de compararlo con Número-Puntos-Ruptura, debe decrementarse para formar el desplazamiento en la tabla de Valores-Sustituidos de ocho bits y posteriormente se desplaza (se multiplica por dos) para formar un desplazamiento en la Tabla-Puntos-Ruptura de 16 bits. El desplazamiento de 8 bits se puede guardar en B y el de 16 bits en A. Las direcciones de las entradas en ambas tablas pueden estar en X e Y, de modo que podemos emplear un autoincremento para recorrer la tabla. La entrada de 16 bits puede ser desplazada a través de U, pero la de 8 bits tendrá que utilizar A de nuevo.

El último proceso empleado en este módulo elimina físicamente un punto de ruptura sustituyendo el opcode SWI con el código original en la Tabla de Valores-Sustituidos.

Eliminar-Punto-Ruptura

Data
Número-Punto-Ruptura es un desplazamiento de 8 bits en la Tabla-Puntos-Ruptura

Proceso:
Tomar el valor que se halla en Valores-Sustituidos (Número-Punto-Ruptura)
Almacenarlo en la dirección indicada en Tabla-Puntos-Ruptura (Número-Punto-Ruptura)

Asumiremos que el parámetro Número-Punto-Ruptura es pasado en B de la forma habitual como número entre 1 y 16, que ha de convertirse para que funcione como un desplazamiento en las tablas.

Estamos ya en la etapa de construir un módulo para ejecutar las ocho órdenes de una letra simple que operan el sistema (véase p. 1238). Varias de estas órdenes pueden ejecutarse por medio de las rutinas que ya hemos escrito. No obstante, para ser completos y obtener una adecuada estructura modular incorporaremos llamadas a dichas rutinas desde este módulo.

La orden B, insertar un punto de ruptura, se consigue plenamente con la rutina Inserción-Puntos-Ruptura (BP01). En este módulo, pues, sólo necesitamos escribir:

```
CMDB BRA BP01
```

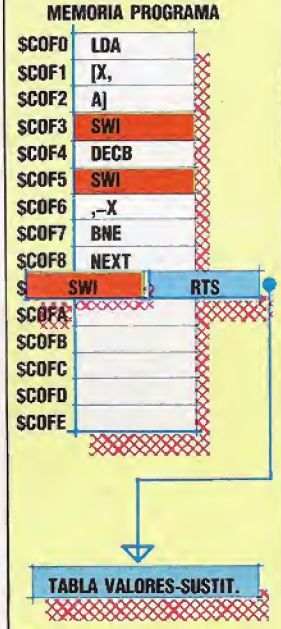
La orden U, desinsertar un punto de ruptura, casi se cumple con la rutina que acabamos de escribir (BP04). Pero primero debemos tomar la dirección del punto de ruptura que hay que descartar buscando en la Tabla-Puntos-Ruptura para encontrarla. Si no se encuentra allí, se ignora la orden; si se encuentra, podemos pasar el desplazamiento a la subrutina en BP02.

La orden U

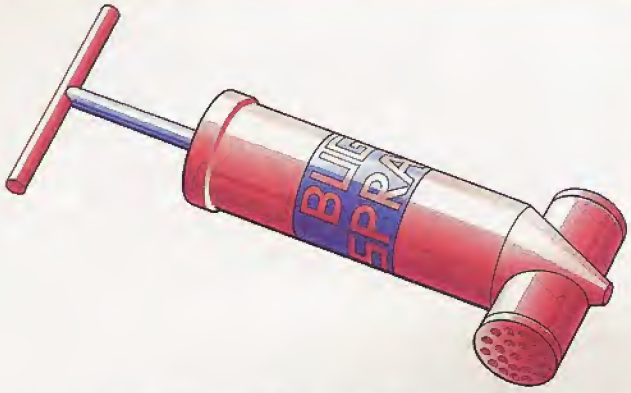
Data:
Aviso a visualizar
Dirección-Punto-Ruptura es la entrada
Tabla-Puntos-Ruptura
Número-Punto-Ruptura

Proceso:
Visualizar el aviso
Tomar Dirección-Punto-Ruptura
Poner a 16 el Número-Punto-Ruptura

Rupturas en el código



El depurador inserta puntos de ruptura en el código objeto bajo comprobación guardando primero el código de esa dirección en la Tabla de Valores-Sustituidos e incrementando el contador de Número-Puntos-Ruptura, para escribir encima del contenido del byte del punto de ruptura por el opcode SWI. La Tabla de Valores-Sustituidos es un hacinamiento de valores que pueden tomarse de cualquier posición. Cuando se quita un punto de ruptura el Valor Sustituido correcto se vuelve a copiar de la tabla en la memoria que contiene el programa, y el byte sobrante se elimina moviendo hacia abajo todos los bytes de la tabla que están encima de él, decrementando, por último, el contador de Número-de-Puntos-Ruptura




```
While (siempre que) Tabla-Puntos-Ruptura
   (Número-Punto-Ruptura) <> Dirección-Punto-
   Ruptura
   y Número-Punto-Ruptura > 0
   Decrementar Número-Punto-Ruptura
Si es encontrada entonces
   Desinsertar-Punto-Ruptura
```

La Dirección-Punto-Ruptura se puede guardar en Y, dejando X disponible para su empleo como índice de la tabla. Número-Punto-Ruptura puede guardarse en B.

La orden D, visualizar los puntos de ruptura, se realiza con la rutina que etiquetamos DISPBP. Se accede a ella con una simple bifurcación a subrutina:

```
CMDD   BRA   DISPBP
```

La orden S, iniciar la ejecución del programa, es algo más complicada, puesto que es aquí donde se insertan los puntos de ruptura. El opcode de la instrucción SWI debe ser insertado en cada dirección dentro de la Tabla-Puntos-Ruptura, y el opcode que ya se encuentra allí se coloca en Valores-Sustituidos. Una vez hecho esto, el control se transferirá a la dirección de inicio del programa. Debemos observar también que el siguiente punto de ruptura es el número 1. He aquí el proceso completo para el inicio del programa:

Orden S

Data:
Número-Puntos-Ruptura es un valor de 8 bits
Tabla-Puntos-Ruptura
Valores-Sustituidos
Número-Punto-Ruptura es un contador de 8 bits
Siguiente-Punto-Ruptura es un valor de 8 bits
Opcode-SWI es un valor de 8 bits
Inicio-Dirección es la dirección de 16 bits inicial del programa que tratamos de depurar
Proceso:
Poner Número-Punto-Ruptura a Número-Puntos-Ruptura
While (siempre que) Número-Punto-Ruptura > 0
Establecer-Punto-Ruptura (Número-Punto-Ruptura)
Decrementar Número-Punto-Ruptura
Endwhile
Poner Siguiente-Punto-Ruptura a 1
Saltar a Inicio-Dirección

Para esto último empleamos la rutina Establecer-Punto-Ruptura, ya codificada, que requiere tener en A el Número-Punto-Ruptura (menos 1, para que pueda servir como desplazamiento en las tablas). Decrementaremos, por conveniencia, a A antes de llamar a la rutina Establecer-Punto-Ruptura. Damos finalmente la rutina codificada.

El modo de concluir esta rutina exige una pequeña aclaración. Cuando se está ejecutando el programa a depurar no se necesitan ítems adicionales en la pila, por lo que habremos de cerciorarnos de que la pila está vacía cuando se transfiere el control al programa. Podríamos borrar todo ítem superfluo de la pila en el nódulo principal, pero si es llamada esta rutina por medio de BSR (para mantener la consistencia con las otras órdenes) la dirección de retorno deberá colocarse en la pila. Y si la dejamos allí en una sesión larga (en la que el programa puede ser reiniciado repetidas veces) la pila puede crecer desmesuradamente. La solución empleada consiste en quitar la dirección de la pila en el momento en que el control es devuelto al programa. Y esto se hace sustituyendo la dirección de retorno colocada en la pila por la dirección inicial. Entonces la RTS hace saltar la dirección de retorno, ya convertida en dirección inicial, fuera de la pila, transfiriendo el control al mismo tiempo que restablece la pila.

En esta lección examinaremos finalmente la orden M, inspeccionar y cambiar posiciones de memoria. Se trata de obtener como entrada una dirección y de visualizar el contenido de esa dirección en la pantalla. El usuario puede entonces introducir un número hexa de dos dígitos para colocarlos en esa posición, o sencillamente un Return. En cualquier caso pasamos a la siguiente posición en la memoria. El usuario puede detener el proceso entrando un punto. La rutina GETHX2 se codificó teniendo esto presente, permitiendo la entrada de dos dígitos hexa o bien de un punto o de un Return.

La orden M

Data:
Posición-Actual es la dirección de 16 bits de la posición que queremos inspeccionar
Valor-Actual se encuentra en Posición-Actual y es de 8 bits
Valor-Nuevo para Posición-Actual; también de 8 bits
Proceso:
Tomar Posición-Actual
Repeat
Visualizar Valor-Actual
Tomar Valor-Nuevo
If Valor-Nuevo no es un punto then
If Valor-Nuevo no es un Return
Almacenar Valor-Nuevo en Posición-Actual
Endif
Incrementar Posición-Actual
Visualizar Posición-Actual
Endif
Until (hasta que) Posición-Actual sea un punto

Para esta rutina se almacenará la Posición-Actual en X, empleándose el registro B tanto para Valor-Actual como para Valor-Nuevo. Por su parte, A sirve de flag que indica cuál de las tres posibilidades (un hexa, un punto o un Return) se ha introducido.

Nos quedan por diseñar y codificar las restantes tres órdenes, G, R y Q. Pero para ello se necesita emplear un mecanismo de interrupción que habremos de analizar detenidamente. De ello trataremos en la próxima lección junto con el diseño del módulo principal del programa depurador de errores.

Rutina Desinsertar

```
BPO4    PSHS    A,B,X,Y,U     Salva regs. empleados
        DEC     NUMBP,PCR     Decrementa Número-Puntos-Ruptura
IF02    CMPB    NUMBP,PCR     Si Entrada-a-Descartar <
        BGT     ENDF02        Número-Puntos-Ruptura
        DECB                  Convierte B en un despl.
        TFR     B,A           Copia B en A
        LSLA                  Convierte A en un despl.
        LDX     BPTAB,PCR     Dirección base de Tabla-Puntos-Ruptura
        LEAX    A,X           Tabla-Puntos-Ruptura (Número-Punto-Ruptura)
        LDY     REMTAB,PCR    Dirección base de Valores-Sustituidos
        LEAY    B,Y           Valores-Sustituidos (Número-Punto-Ruptura)
FOR00   LDU     2,X           Toma la entrada de Tabla-Puntos-Ruptura a mover
        STU     ,X++          La mueve un lugar hacia atrás
        LDA     1,Y           Toma la entrada de Valores-Sustituidos que hay que mover
        STA     ,Y+           La mueve un lugar hacia atrás
        INCB
        CMPB    NUMBP,PCR     ¿Ultima?
        BLT     FOR00         La siguiente
ENDF02  PULS    A,B,X,Y,U,PC  Restaurar y retornar
```

Rutina Eliminar-Punto-Ruptura

```
BPO5    PSHS    A,B,X
        DECB                  Conversión en despl. para Valores-Sustituidos
        LDX     REMTAB,PCR    Dirección base de Valores-Sustituidos
        LDA     B,X           Toma el valor a mover
        LSLB                  Conversión en despl. para Tabla-Puntos-Ruptura
        LDX     BPTAB PCR     Dir. base de Tabla-Puntos-Ruptura
        STA     [B,X]         Almac. en tabla según dir.
        PULS    A,B,X,PC      Restaurar y retornar
```

La orden U

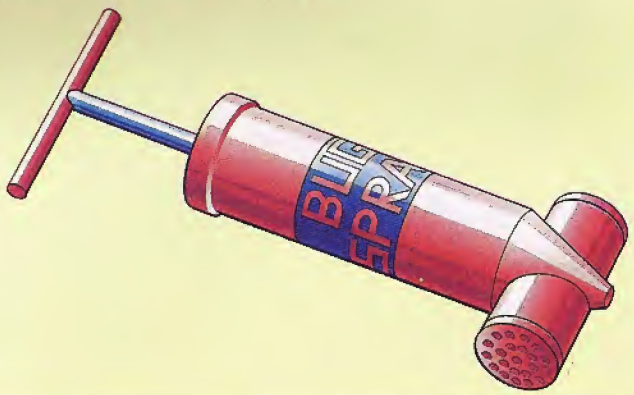
```
PROMPT  FCB     '>
CMDU    PSHS    A,B,X,Y       Guarda regs. empleados
        LDA     PROMPT,PCR
        BSR     OUTCH         Visualiza el aviso
        BSR     GETADD        Toma Dirección
        TFR     D,Y           Pone Dir.-Punto-Ruptura en Y
        LDB     MAXBP,PCR     Número máximo de Puntos-Ruptura (16)
        LDX     BPTAB,PCR     Dir. base de Tabla-Puntos-Rupt.
        TFR     B,A
        LSLA                  A es un desplazamiento al final de Tabla-Puntos-Ruptura
        LEAX    A,X           X señala ahora más allá del final de la tabla
        TSTB                  Activa los flags según contenido de B
WHIL02  BLE     ENDW02        While B>0
        CMPY    ,--X          (Recuerde que X se decrementa primero)
        BEQ     ENDW02        y (and) Dir.-Puntos-Rupt. no se encuentra en la tabla
        DECB                  Decrementa Número-Punto-Ruptura
        BRA     WHIL02
ENDW02  TSTB                  Se encuentra si B>0
IF03    BLE     ENDF          Si se encuentra, entonces
        BSR     BP04          Desinsertar-Punto-Ruptura
ENDF03  PULS    A,B,X,Y
```

La orden S

```
START   RMB     2             Dirección-Inicio
CMDS    LDA     NUMBP,PCR     Pone Número-Punto-Ruptura a Número-Puntos-Ruptura
WHIL03  TSTA                  Prueba el valor de Número-Punto-Ruptura
        BLE     ENDW03        While Núm.-Punto-Rupt.>0
        DECA                  Decrementa Núm.-Punto-Rupt.
        BSR     BP02          Establecer-Punto-Ruptura
        BRA     WHIL03        Siguiente-Punto-Ruptura
ENDW03  LDA     # 1           Poner a 1 Siguiente-Punto-Ruptura
        STA     NEXTBP,PCR
        STD     1,S
        RTS
```

La orden M

```
PROMPT  FCB     '>
SPACE   FCB     32            Espacio, en código ASCII
CMDM    PSHS    A,B,X         Guarda regs. empleados
        LDA     PROMPT,PCR
        BSR     OUTCH         Visualiza el aviso
        BSR     GETADD        Toma Posición-Actual
        TFR     D,X           La lleva a X
REPT01  LDB     ,X            Toma Valor-Actual
        BSR     PUTHEX        Lo visualiza
        LDA     SPACE,PCR
        BSR     OUTCH         Visualiza un Espacio
        BSR     GETVAL        Toma Nuevo-Valor
IF03    TSTA                  Si Valor-Nuevo no es un punto
        BLT     UNTL01
        BGT     ENDF03        Si no es un Return
        STB     ,X            Almacena Valor-Nuevo en Posición-Actual
ENDF03  LEAX    1,X           Incrementa Posición-Actual
        TFR     X,D           Visualiza Posición-Actual
        BSR     DSPADD
        BRA     REPT
UNTL01  PULS    A,B,X,PC
```



La suma de las partes

He aquí un repaso de las materias que hemos estudiado hasta el momento en este apartado

Los micros BBC y Commodore 64 poseen una disposición de entrada/salida similar que permite la comunicación con el mundo exterior a través de una puerta para el usuario, que consiste esencialmente en ocho patillas de datos y una conexión a tierra. Cada una de estas ocho patillas de datos está asociada a una posición particular de la memoria, denominada *registro de datos*, correspondiendo cada patilla a un bit en el registro. Una segunda posición, el registro de dirección de datos (RDD), controla la dirección de los datos que fluyen a o desde cada patilla. Si cualquiera de las patillas se establece en salida (bit RDD=1), en la patilla se induce un voltaje de + 5 V cada vez que el bit correspondiente se establece alto (en uno). Si el bit de datos se establece bajo, en la patilla se induce un voltaje de 0 V. Si bien la corriente que se suministra desde las patillas de datos de la puerta para el usuario no puede activar directamente dispositivos externos, sí se la puede utilizar para disparar un sistema de relé que permita encender o apagar voltajes mayores y/o sistemas que estén conectados a la red eléctrica.

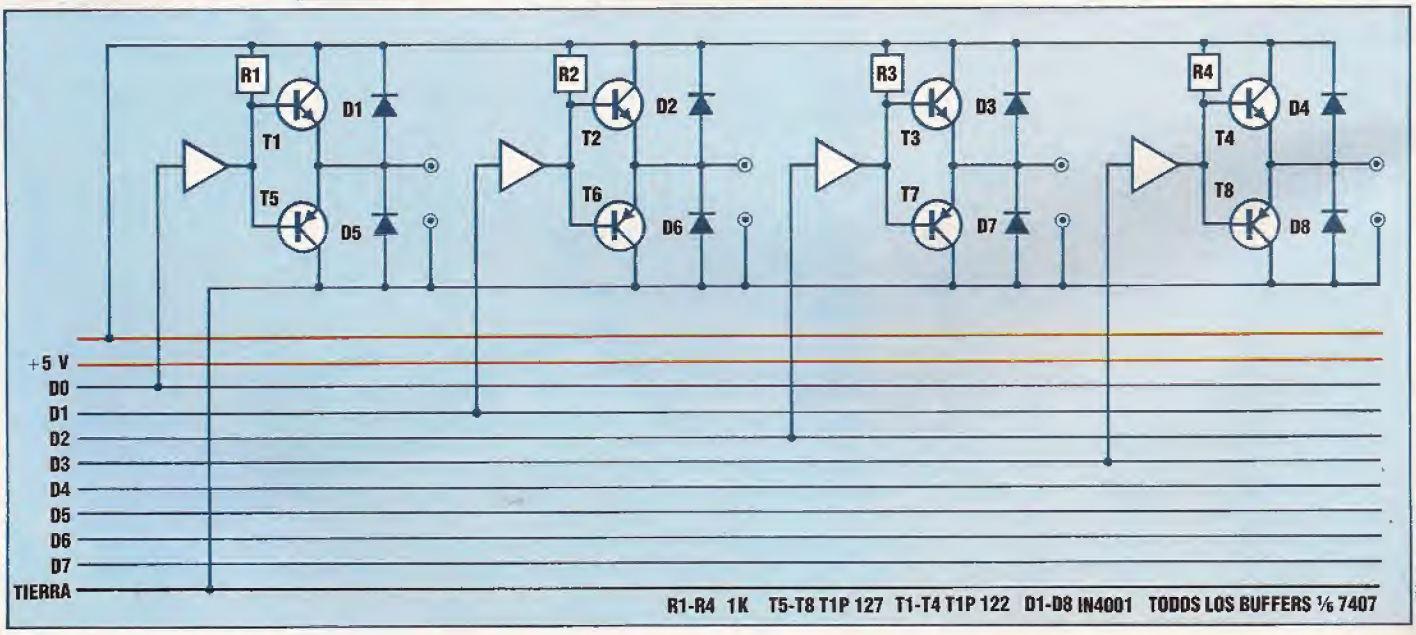
Cuando se establece una patilla para entrada (bit RDD=0), entonces el método de operación es bastante diferente. En este caso, el bit correspondiente del registro de datos se mantiene alto, bajando solamente si la patilla se conecta a tierra. Esto se puede emplear para controlar eventos del mundo exterior, conectando un lado de un sencillo interruptor a una patilla de datos y el otro lado a la toma de tierra de la puerta para el usuario. Cuando se mueve el interruptor, la patilla de datos conecta a tierra y el bit correspondiente del registro de datos sufre una transición de alto a bajo.

Las ocho líneas de datos y la toma de tierra se deben conectar de alguna forma a cada dispositivo del sistema de la puerta para el usuario y, de este modo, todo el sistema completo se designa alrededor de un bus común de nueve líneas, conectado cada dispositivo en la línea adecuada para su función particular. Este bus común se conecta a cada dispositivo mediante un conector minicon de 12 vías. Conectando a un conector macho del lado "interior" y un conector hembra del lado "exterior" en cada dispositivo, podemos realizar una "cadena margarita" de cualquier combinación de componentes del sistema entre sí.

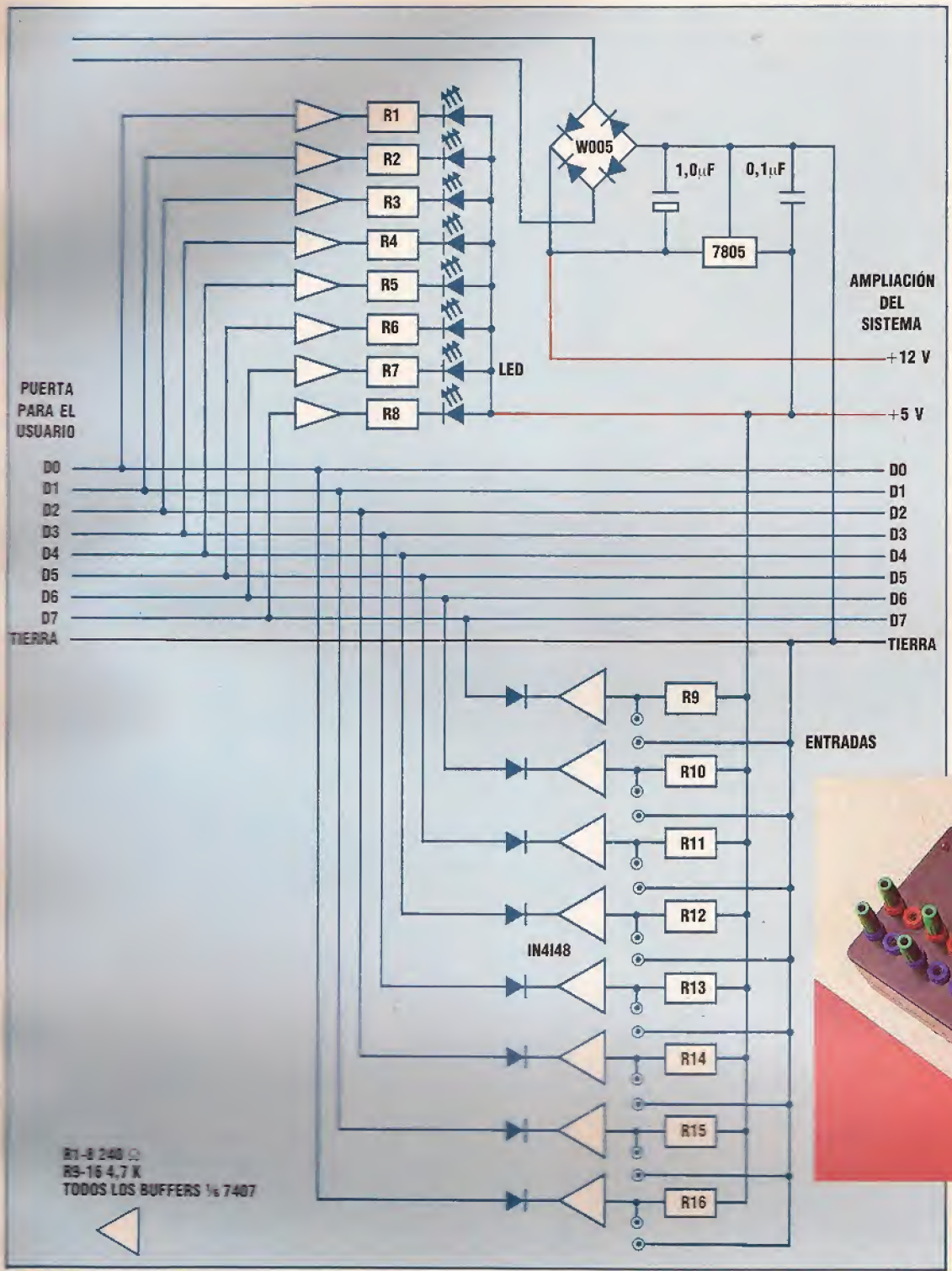
Ian McKinnell

La caja de salida

La caja de salida de bajo voltaje se conecta con el bus del sistema a través de un conector macho minicon de 12 vías, que se enchufa en su equivalente hembra de la caja buffer. La corriente suministrada por una patilla de datos activada es del orden de unos pocos miliamperios, insuficiente para activar un dispositivo como, por ejemplo, un motor eléctrico, pero suficiente para actuar a modo de corriente de conmutación a través de un transistor. Las líneas de datos de la 0 a la 3 las utiliza la caja de bajo voltaje. La activación (estado "alto") de una de estas líneas hace que el voltaje del transformador se conmute a través de un transistor al correspondiente conector de red de esta caja. Por lo tanto, con el voltaje de entrada se puede abastecer simultáneamente a cuatro dispositivos.



Liz Dixon

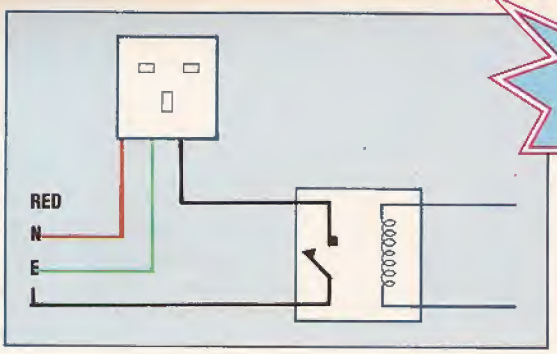


La caja buffer

Es el dispositivo más importante del sistema de la puerta para el usuario. El sistema de circuitos protege los chips de E/S del ordenador contra cualquier intento de "sorber" demasiada corriente de una patilla de datos o por aplicar un voltaje de entrada peligroso. Además, acepta y regula una entrada de CD a CA desde un transformador, en la escala de 5 a 21 V. Esta entrada de voltaje se agrega al bus del sistema como un par de líneas extras, para utilizar con otros componentes del sistema. Con la caja buffer conectada se pueden efectuar entradas en la puerta para el usuario; los ocho conectores rojos corresponden a las ocho líneas de datos, los conectores negros proporcionan una toma a tierra separada para cada línea de datos. Hay montada en la caja una serie de ocho LED. Cada LED se ilumina si su línea de datos se desactiva

El relé de red

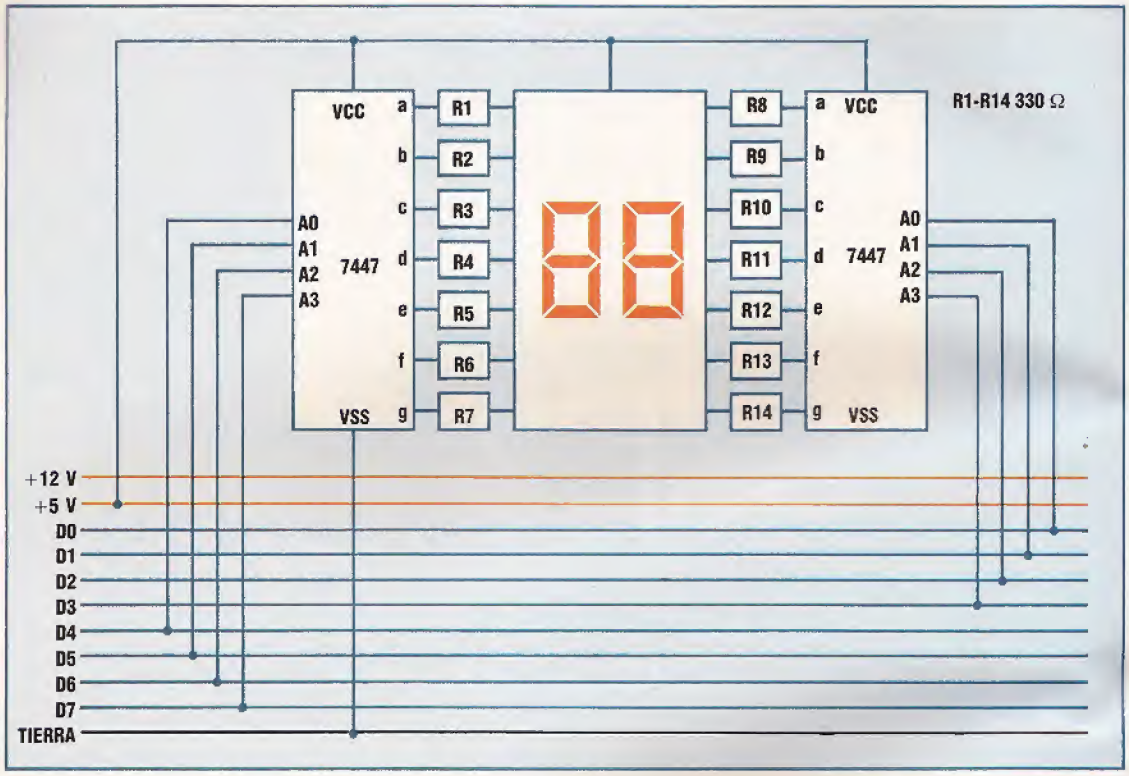
Se puede hacer que el voltaje del transformador conmute un voltaje de la red mediante un relé. Esta unidad se enchufa a la red y a una de las cuatro líneas de la caja de salida. Al establecer alto uno de los bits del registro de datos, se conmuta la alimentación del transformador al conector de salida correspondiente, que a su vez conmuta la alimentación de la red al conector de tres vías. Así es posible controlar aparatos eléctricos desde el ordenador. (Véase p. 1126.)



¡ATENCIÓN!
Todo lo que implica potencia eléctrica exige sumo cuidado.
- **Antes de comenzar a trabajar desconecte las fuentes de potencia.**
- **Compruebe conexiones y aislamientos mediante un tester.**
- **Evite los cortocircuitos.**

Visualización en siete segmentos

Requiere 4 entradas lógicas para visualizar los 16 dígitos hexa, de modo que utilizando las ocho líneas de entrada disponibles en el bus del sistema podemos activar dos de tales visualizaciones. Esta unidad visualizará, por consiguiente, el contenido del registro de datos de la puerta para el usuario como un par de dígitos hexa, y se puede conectar ya sea a la caja buffer o bien a un conector minicon hembra de la caja de salida

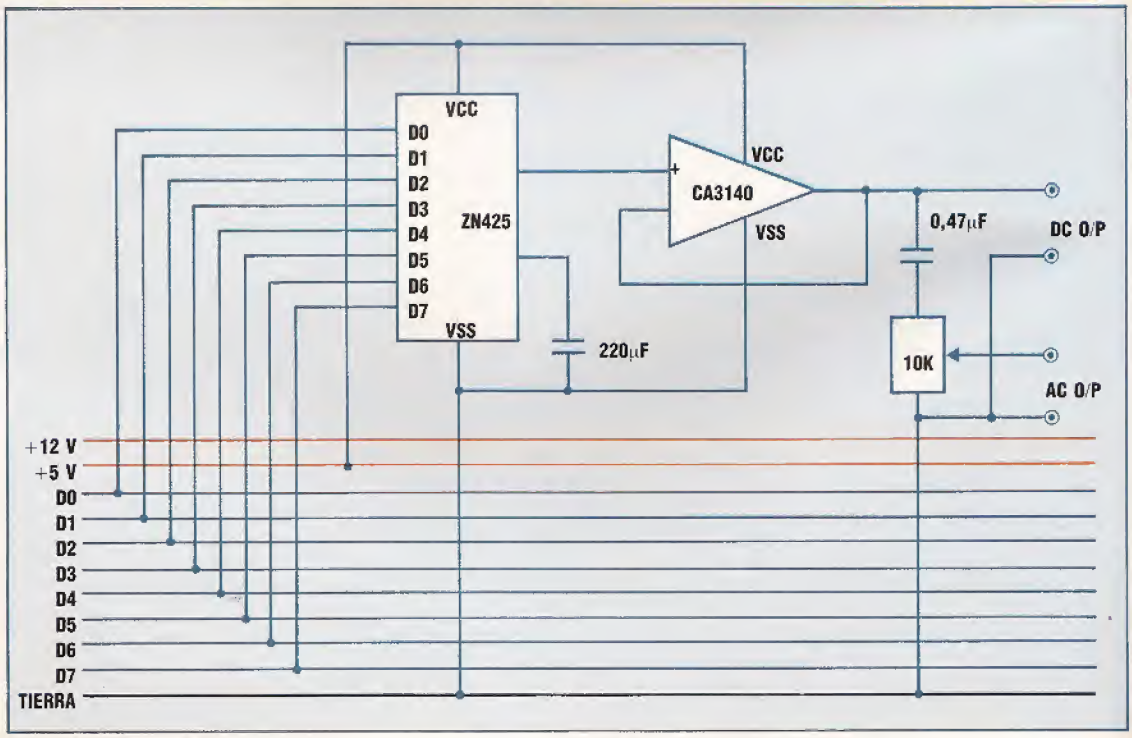


Kevin Jones

El convertidor D/A

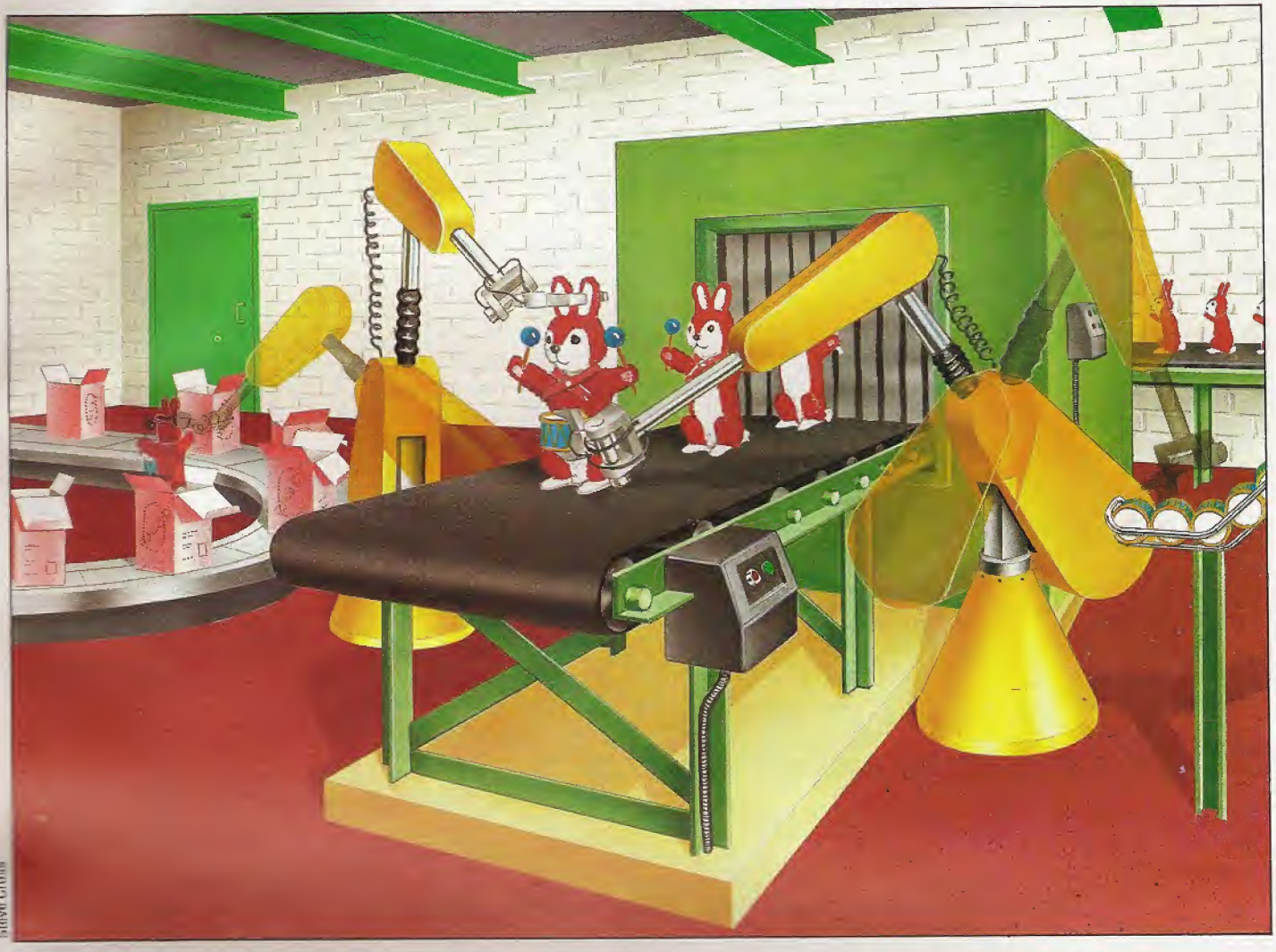
El convertidor de digital a analógico convierte un valor del registro de datos entre 0 y 255 en un voltaje. La salida de la caja es de alrededor de 600 W, no lo suficientemente fuerte como para activar directamente motores u otros dispositivos de gran consumo, pero se la puede amplificar para hacerlo. Por otra parte, el convertidor se puede utilizar para generar sonido, ya sea a través de auriculares o bien de un amplificador de audio

Ian McKinnell



Réplica perfecta

La mejor forma de "trabajar" con un robot en casa es mediante la creación de un modelo del mismo por ordenador



Lloyd Cross

El desarrollo de la tecnología del ordenador ha llevado a una utilización cada vez mayor de las simulaciones: se pueden construir "modelos" por ordenador que imiten fidedignamente acontecimientos del mundo real. La mayoría de las personas ya están familiarizadas con el concepto de simuladores de vuelo, dispositivos sumamente complejos que permiten a los aspirantes a piloto adquirir experiencia de vuelo sin tener que pilotar un avión auténtico. Pero hay muchas otras actividades que también se prestan a la simulación por ordenador: previsiones comerciales, operaciones de ingeniería y procesos físicos de todo tipo se pueden simular muy fácilmente en un modelo por ordenador. En algunos casos, éste puede llevar a cabo experimentos que, de realizarse por algún otro medio, serían demasiado peligrosos. Por ejemplo, es de vital importancia descubrir lo que sucedería en una central de energía nuclear si se produjera una fuga de líquido refrigerante del reactor. En este caso, obviamente sería imposible utilizar una central nuclear auténtica para llevar a cabo el experimento, de modo que se emplea una simulación por ordenador. Si el modelo es suficientemente detallado, entonces se puede ver con exactitud lo que sucedería si se produjera la fuga.

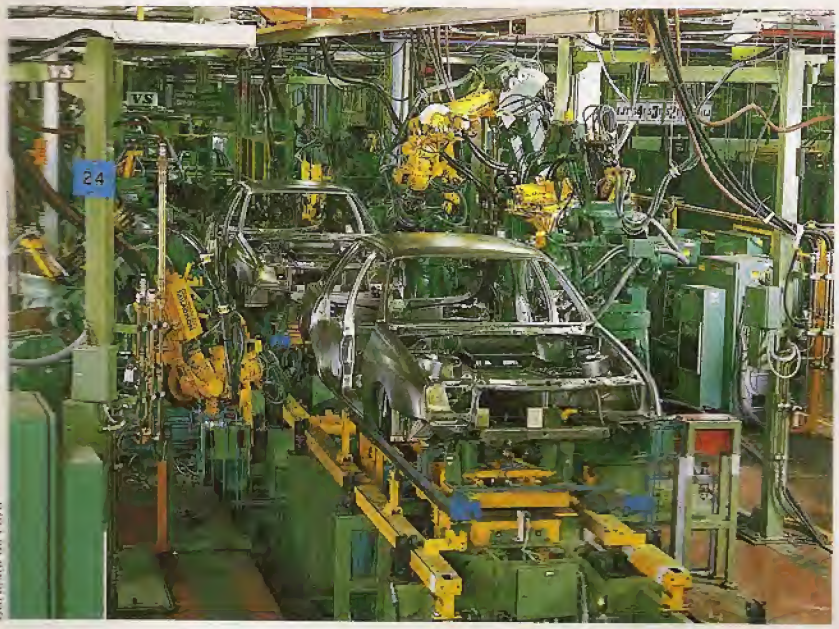
Del mismo modo, en el campo de la robótica se utilizan simulaciones por ordenador para diseñar nuevos robots. Es evidente que se puede proceder mediante ensayo y error (construir un robot, observar cómo se comporta y luego efectuar todas las modificaciones necesarias), pero esto es costoso y lleva mucho tiempo. Una simulación por ordenador permite diseñar un robot y controlar sus acciones sin costo alguno y sin el esfuerzo físico que implica realizar frecuentes cambios de diseño.

Tomemos como ejemplo una cadena de montaje de automóviles, en la cual un equipo de robots va

Movimientos armónicos
Cuando los robots realizan una tarea conjunta existe una auténtica necesidad de coreografía, con el objeto de que no interfieran entre sí. Aquí, un brazo puede recoger y mantener el juguete en posición mientras el otro robot recoge un tambor y lo coloca en su sitio; el primer brazo coloca entonces el juguete ya completo en su caja. Si el movimiento del brazo se controla adecuadamente, las cintas transportadoras se pueden ubicar en cualquier disposición que resulte mejor para el resto de la cadena de montaje, mientras que las exigencias ergonómicas de un operador humano limitarían esta libertad

trabajando en los coches a medida que van pasando. Todo lo que se desea es programar los robots de modo que ensamblen los vehículos de la forma correcta. No obstante, programar los robots ocupa tiempo y, cada vez que se paraliza la cadena de montaje, se pierde dinero. Podría optarse por crear una línea de montaje falsa con algunos robots totalmente diferentes con los cuales desarrollar los nuevos programas. Pero esto también es caro y puede conducir fácilmente a otra dificultad: el problema de la coreografía de los robots, de la cual ya hemos hablado anteriormente. Es de vital importancia asegurar que los robots que estén trabajando juntos no interfieran mutuamente en sus movimientos. Esto no es sólo una cuestión de conveniencia: un gran robot industrial capaz de transportar cargas pesadas podría con toda facilidad dañar a otro robot si tropezara con él. Y no sólo los robots pueden sufrir daños: ¡un robot programado incorrectamente que se pasara el tiempo soldando puertas de coches para cerrarlas enseguida sería desastroso!

Perfecta coordinación
La complejidad del movimiento y la sincronización que requieren las aplicaciones reales de la robótica se pueden apreciar claramente en esta sección de montaje del Ford Sierra



Cortesía de Ford

La respuesta obvia no es otra que llevar a cabo simulaciones por ordenador de las acciones de cada robot, de modo que el usuario pueda ver cómo interactúan las mismas. De esta forma los costos son bajos y no hay nada que pueda resultar dañado. Una vez que la simulación se haya completado y cuando todo parezca ser satisfactorio, se pueden transferir fácilmente los programas así desarrollados a robots reales, que entonces ya pueden llevar a cabo con total seguridad las tareas asignadas.

En este capítulo demostraremos el principio de la simulación por ordenador mediante un programa, *Brazo-robot*, que simula un brazo-robot de "coger y colocar" con dos grados de libertad. No posee sensores, de modo que usted mismo debe guiarlo, controlando las junturas del hombro y del codo y el mecanismo de prensión del efector final, con el fin de asir un objeto y colocarlo luego en algún otro lugar. Adicionalmente, remítase al programa para resolver laberintos que ofrecimos en la página 1202, que muestra cómo se puede programar un robot para que encuentre su camino hasta el centro de un laberinto. Este programa es, en realidad, una simulación por ordenador de cómo intentaría alcanzar su objetivo un robot real. Imita las acciones de un autómata cibernético dotado de un sencillo sensor táctil y encuentra la ruta correcta simplemente avanzando hacia espacios vacíos hasta hallar un callejón sin salida, punto en el cual retorna a la última intersección que había atravesado y trata entonces de seguir avanzando por un nuevo sendero. Este modelo no tiene prácticamente ninguna sofisticación, pero ilustra cómo se puede utilizar un programa de ordenador para simular los movimientos de un robot. El robot del programa obedece un conjunto estipulado de reglas y «traza un mapa» del entorno. Si, dentro del programa, tuviera acceso directo e inmediato a las posiciones de todos los componentes del laberinto, sería capaz de avanzar directamente hasta su meta. En nuestro programa el robot carece de esta información y, por tanto, debe valerse de una técnica de ensayo y error.

De forma similar, el programa *Brazo-robot* imita el comportamiento de un robot que no posee ningún sensor. Este programa contiene un modelo del entorno del robot y otro del brazo propiamente dicho, y es necesario asegurarse de que ambos modelos interactúan sólo como lo harían en la vida real. Por consiguiente, no se puede recoger un objeto con el brazo a menos que el mismo esté posicionado correctamente. Y tampoco es posible desplazar el brazo por debajo del nivel del suelo, dado que ello sería imposible si el brazo-robot fuera auténtico. A pesar de que estamos utilizando gráficos por ordenador, en los cuales una línea (que representa el brazo) podría fácilmente cruzarse con otra (el suelo), para una simulación exacta es preciso que estas líneas no se crucen. Y cuando el robot suelta un objeto, éste no debe permanecer necesariamente en esa posición: su simulación debe permitir efectos gravitacionales. Si este punto se ignorara, ¡ciertamente no se podría desarrollar una simulación segura para que un robot de "coger y colocar" manipulara huevos!

Para obtener mayor realismo

Existen muy pocas limitaciones en cuanto a lo que se puede conseguir utilizando la simulación por ordenador; y, en la mayoría de los casos, cuanto más compleja es la simulación, más fascinante resulta. Tal simulación puede ser aún más entretenida que el simple hecho de jugar con robots reales, por la sencilla razón de que, empleando una simulación, uno puede diseñar el robot que desee; la programación de los detalles correctos de éste y su mundo puede conducir a una mejor comprensión de los robots y de la forma que funciona el mundo físico. Observe otra vez el programa *Brazo-robot*. Verá que cuando el robot deja un objeto, éste cae al suelo y permanece allí. Para lograr que el modelo sea aún más realista, se puede modificar el programa de modo que el objeto vaya acelerando a medida que cae, obedeciendo, por lo tanto, la ley de gravedad. Y, tal vez, ¿podría incluso rebotar al caer al suelo?

Las posibilidades son muchas y el programa está allí, para que usted lo adapte, añadiéndole características nuevas y más realistas para que su simulación sea lo más "viva" posible.

```
  4 REM "******SPECTRUM**********
  5 REM " SIMULACION BRAZO ROBOT "
  6 REM "******SPECTRUM**********
 10 CLS: PRINT "BRAZO ROBOT": PRINT: PRINT "LOS MANDOS PARA
    EL BRAZO ROBOT SON:"
 15 PRINT "S- SELECCION ROTACION HOMBRO": PRINT "E-
    SELECCION ROTACION CODO": PRINT "K- ROTACION JUNTURA
    SENTIDO HORARIO"
 20 PRINT "H- ROTACION JUNTURA SENTIDO ANTI-HORARIO":PRINT
    "U- COGER BOLA": PRINT "I- SOLTAR BOLA"
 25 PRINT AT 20,0;"PULSE una tecla":PAUSE 0:
    RANDOMIZE
1000 GO SUB 9600      REM inicio
1100 GO SUB 9100      REM entrada
1200 GO SUB 9000      REM calca
1400 STOP
1500 REM " actualizar juntura/s **********
1550 LET ex="COS hd1: LET ey=l1*SIN hd1
1580 LET hx=sx-ex:LET hy=sy+ey
1650 LET wx=l2*SIN hd2: LET wy=l2*COS hd2
1680 LET hx=hx+wx:LET hy=hy+wy
1690 RETURN
2000 REM " dibujar brazo **********
2020 INK acol
2050 PLOT INVERSE rubout;sx,sy
2100 DRAW INVERSE rubout;ex,ey: DRAW INVERSE rubout;wx,wy
     : IF blup THEN LET br=FN r(hy): LET bc=FN c(hx):
     GO SUB 2500
2490 RETURN
2500 REM " dibujar bola **********
2510 INK bcol
2550 PRINT AT br,bc;" ■ "(rubout+1)
2740 RETURN
2750 REM " soltar bola **********
2800 LET rubout=1: GO SUB 2000: LET rubout=0
2820 LET k=INT (xh*RND): IF k>=xs THEN IF k<=xs+wd THEN GO TO
     2820
2850 LET br=FN r(y0+4): LET bc=FN c(k): LET blup=0: GO SUB 2000:
     GO SUB 2500
2890 RETURN
2900 REM "      rotar      **********
2910 LET rubout=1: GO SUB 2000
2920 LET l1=dirn*sr*a1: LET t2=t1+dirn*er*a2
2950 LET hd1=hd1+t1: LET hd2=hd2+t2
2990 GO SUB 1500
3000 IF ABS hd1>p2 THEN LET ok=0
3020 LET pt=POINT (hx,hy)
3040 IF pt<>0 THEN LET ok=0: IF br=FN r(hy) AND bc=FN c(hx) THEN
     LET ok=2
3400 LET rubout=0:GO SUB 2000
3450 INK pacol:PRINT AT 21,0;s$;AT 21,2;FN d(hd1);AT 21,26;
     FN d(hd2)
3490 RETURN
5000 REM "      entrada **********
5100 IF INKEY$<>"" THEN GO TO 5100
5120 FOR i=0 TO 1 STEP 0
5150 LET a$=INKEY$: IF a$>="A" AND a$<="Z" THEN LET
     a$=CHR$(CODE a$+32)
5200 IF a$="s" THEN LET sr=1:LET er=0
5220 IF a$="e" THEN LET er=1:LET sr=0
5250 IF a$="k" THEN LET dirn=1:GO SUB 3000
5270 IF a$="h" THEN LET dirn=-1:GO SUB 3000
5300 IF a$="u" THEN IF ok=2 THEN LET blup=1
5320 IF a$="i" THEN IF blup THEN GO SUB 2750
5400 IF ok=0 THEN LET i=2
5490 NEXT i
5495 RETURN
5500 REM "      calca **********
5510 PAPER 6: CLS
5520 INK 0
5530 FOR i=0 TO 21:PRINT AT i,0;CHR$ 144;
5540 NEXT i
5550 INK 7: GO SUB 2500: INK acol
5600 PRINT AT 20,2;"HOMBRO";AT 20,26;"CODO"
5690 RETURN
9000 REM "      inicio **********
9050 DEF FN s(x)=INT (x*180/PI)
9060 DEF FN r(x)=21-INT(x/8)
9070 DEF FN c(x)=INT(x/8)
9080 DIM s$(32):LET xl=0:LET yl=0:LET xh=
     254:LET yh=174
9120 LET y0=23:LET wd=60:LET ht=23
9130 LET blup=0:LET bc=2:LET br=FN r(y0+4)
9140 LET grcol=3:LET bacol=2:LET acol=1:LET bcol=6:
     LET pacol=7
9150 LET sx=xh/2:LET sy=y0+ht+2:LET l1=(yh-ht-y0-2)/2:IF
     l1>xh/4 THEN LET l1=xh/4
9160 LET l2=l1:LET hx=0:LET hy=0
9170 LET p2=PI/2:LET a1=PI/32:LET a2=2*a1
9180 LET hd1=0:LET hd2=p2
9190 LET sr=1:LET er=0:LET dirn=1:LET rubout=
     0:LET ok=1
9750 GO SUB 9000:GO SUB 1500:GO SUB 2000
9790 RETURN
```

Coger y colocar

Este programa simula un brazo-robot que puede estirarse hasta alcanzar un objeto, asirlo y colocarlo luego en otro lugar. Su tarea consiste simplemente en recoger la bola y luego dejarla caer. El brazo está diseñado para utilizar coordenadas de revolución con dos grados de libertad: una juntura de hombro y una juntura de codo. La primera puede rotar a través de 180°; la segunda a través de 360°.

El programa se controla mediante las siguientes teclas: S indica que se desea un movimiento de hombro; E un movimiento de codo; las teclas K y H harán que la juntura del brazo rote en el sentido de las agujas del reloj o en el sentido contrario, respectivamente. Cada pulsación hace rotar la juntura del hombro en 6°, y en 12° la juntura del codo. U indica que se desea que el brazo intente asir la bola. Esto sólo se realizará con éxito si se ha logrado manipular el brazo dentro del alcance de la misma. F hará que el robot la deje caer

Ian McKinnell

Mover y recoger
Nuestro programa de simulación de un brazo-robot permite mover un brazo de dos junturas en dos dimensiones y recoger un objeto con el mismo. Cuando se deja caer el objeto, el programa lo coloca al azar sobre el suelo. La visualización muestra los ángulos verticales formados por los brazos superior e inferior

Para el BBC Micro realice las siguientes añadiduras y modificaciones:

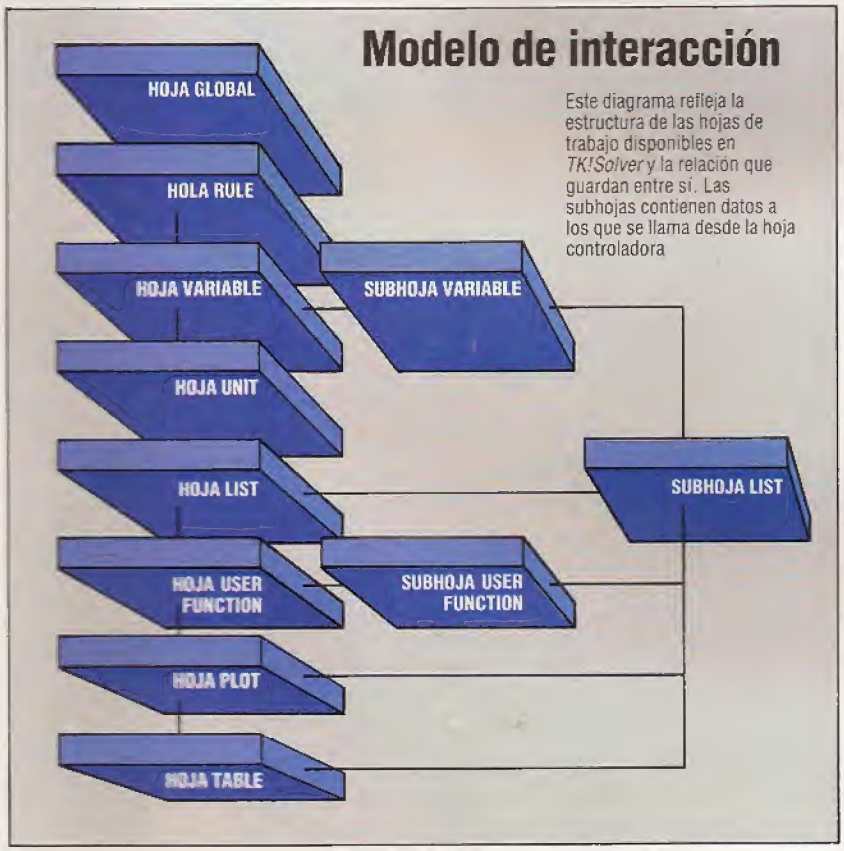
```
  4 REM ****************BBC ****************
  5 REM*          SIMULACION BRAZO ROBOT          *
  6 REM ****************BBC ****************
  7 MODE 1:COLOUR 130:COLOUR 1:CLS
 25 PRINTTAB(15,20)"PULSE UNA TECLA":A$=GET$
1000 GOSUB 9600
2020 GCOL 0,acol
2050 MOVE sx,sy
2100 PLOT rubout,ex,ey:PLOT rubout,wx,wy:IF blup THEN
     br=hy:bc=hx:GOSUB 2500
2190 RETURN
2200 REM********** COGER LA BOLA **********
2250 blup=1:rubout=3:GOSUB 2500
2300 rubout=1:GOSUB 2000
2600 GCOL 0,bcol:MOVE bc, br
2650 PLOT 0,0,bsz:PLOT 80+rubout,bsz,0
2700 PLOT 0,0,-bsz:PLOT 80+rubout,-bsz,0
2800 rubout=3:GOSUB 2000:rubout=1
2820 k=INT(xh*RND):IF k>=xs THEN IF k<=xs+wd THEN GOTO
     2820
2850 br=y0-5:bc=k:blup=0:GOSUB 2000:GOSUB 2500
2900 rubout=3:GOSUB 2000
2940 IF pt<>pacol-128 THEN ok=0:IF pt=bcol THEN ok=2
3400 rubout=1:GOSUB 2000
3450 COLOUR pacol:PRINTTAB(0,3)s$;TAB(4,3);FNd
     (hd1);TAB(27,3);FNd(hd2)
5100 IF INKEY$(0)<>"" THEN GOTO 5100
5150 a$=INKEY$(0):IF a$>="A"ANDa$<="Z"THENa$=
     CHR$(ASC(a$)+32)
5300 IF a$="u" AND ok=2 THEN GOSUB 2200
5400 IF ok=0 THEN i=2
6100 PRINTTAB(12,3)"!!CRASH!!":SOUND 1,-15,48,10:SOUND
     1,-15,4,20:RETURN
9050 GCOL 0,pacol:COLOUR pacol:CLS
9100 GCOL 0, grcol
9120 FOR k=0 TO y0:MOVE 0,k:DRAW xh,k:NEXT k
9200 GCOL 0,bacol:xs=(xh-wd)/2
9240 MOVE xs,k:DRAW xs+wd,k
9300 MOVE bc,br:GOSUB 2500:COLOUR acol
9400 PRINTTAB(2,2)"HOMBRO":TAB(26,2)"CODO"
9600 s$="  ":x1=0:y1=0:xh=1000:yh=1000
9620 y0=100:wd=200:ht=100
9630 blup=0:bsz=wd/5:bc=40:br=y0+5
9640 grcol=3:bacol=2:acol=2:bcol=0:pacol=129
9650 sx=xh/2:sy=y0+ht+2:l1=(yh-ht-y0-2)/2:IF l1>xh/4 THEN
     l1=xh/4
9690 sr=1:er=0:dirn=1:rubout=1:ok=1
```

Solución avanzada

"TK!Solver" aporta una nueva dimensión a la hoja electrónica: el proceso de ecuaciones

Como ya hemos visto en este apartado, los programas de hoja electrónica para microordenadores pueden ser muy útiles para una gran variedad de tareas matemáticas. Para la persona acostumbrada a trabajar con grandes hojas compuestas de filas y columnas, con un lápiz y una calculadora, la hoja electrónica representa un valioso elemento que ayuda a ahorrar tiempo y esfuerzo. No obstante, las hojas electrónicas poseen limitaciones significativas. El formato de fila y columna, ideal para contabilidad u otros modelos financieros, a menudo es incómodo y, en ocasiones, inútil para aplicaciones científicas y matemáticas de nivel más elevado. Y las hojas electrónicas tienen una estructura muy rígida para manipular ecuaciones.

Software Arts, la empresa norteamericana que creó el *VisiCalc*, ha desarrollado un programa llamado *TK!Solver*, que va más allá de las hojas electrónicas tanto en su forma como en sus funciones. "*TK!*" alude a *ToolKit* (juego de herramientas), mientras que "*Solver*" (el que soluciona) es la sección de código que procesa ecuaciones. Además de ser diferente a las hojas electrónicas desde el punto de vista del formato de la pantalla, *TK!* ofrece las siguientes características exclusivas:



Liz Dixon

Backsolving: Las fórmulas de hoja electrónica sólo pueden resolver una única variable. *TK!* puede resolver cualquier variable de una ecuación si se le proporcionan datos suficientes para hacerlo;
Iteración: Si falta o no se conoce algún valor necesario para resolver una ecuación, se puede entrar un valor de ensayo y *TK!* lo utilizará como punto de partida. Luego resolverá la ecuación a través de una serie de aproximaciones sucesivas;
Conversión de unidades: Puede convertir valores de pies a metros, de dólares a libras, etc., instantáneamente a partir de tablas de conversión;
Funciones matemáticas: Tiene incorporadas una enorme variedad de ellas.

Hojas de trabajo

TK!Solver opera a través de tres hojas de trabajo enlazadas, cada una de ellas con una función específica. La hoja Variable contiene los nombres de todas las variables definidas, columnas para los valores que entra el usuario y los valores que produce el programa, un lugar para indicar unidades que guardan alguna relación, y un espacio para que el usuario anote un comentario para cada variable. La hoja Variable aparece en la parte superior de la pantalla que visualiza inicialmente el programa. Cada variable también se escribe en detalle en una subhoja de variables separada. La hoja Rule se utiliza para entrar las ecuaciones que *TK!* ha de resolver. Una ecuación puede tener hasta 200 caracteres de longitud, y debe responder a las convenciones matemáticas estándares en cuanto a notación y operaciones. La hoja Rule ocupa la porción inferior de la pantalla inicial del programa. La hoja Unit almacena la información necesaria para convertir las unidades de medida que acompañan a las variables de un modelo.

TK! emplea estas tres hojas para llevar a cabo la mayoría de sus operaciones. Entre las otras se incluyen la hoja Global, en la que el usuario puede construir algunos de los procedimientos operativos del programa; la hoja List, que almacena una matriz de valores para variables; la hoja User Function, para funciones definidas por el usuario; y hojas para trazar o imprimir puntos o tablas de valores.

Creación de un modelo

Empezaremos por crear un modelo muy simple adaptado del manual de usuario del *TK!*, que calcula los gastos del recorrido y la velocidad media para un viaje en automóvil, y convierte los valores de unidades inglesas a unidades métricas. En la visualización inicial encontramos el cursor en la hoja Rule, en la parte inferior de la pantalla. Comenza-

mos por definir las variables en ecuaciones apropiadas, de modo que tecleamos:

distancia/tiempo=velocidad

y pulsamos Return. Inicialmente, el programa está preparado para leer nombres de variables de ecuaciones directamente en la hoja Variable de la parte superior. *TK!* evalúa la ecuación y visualiza las variables en la columna Name de la hoja Variable por el mismo orden en que aparecen en la ecuación. Luego se visualiza un asterisco en la columna Status (estado) al lado de la ecuación. El asterisco significa que en la hoja de variables no se ha entrado ningún valor. Luego entramos la segunda ecuación por el mismo procedimiento:

distancia/gasolina=gastosviaje

Ecuaciones y variables

Después de entrar esta ecuación, las cinco variables definidas se listarán en la columna Name de la hoja Variable.

Pulse la tecla del punto y coma (;) para mover el cursor desde la hoja Rule hasta la hoja Variable, en la cual ya podemos entrar valores. El cursor aparece en la columna de entradas junto a nuestra primera variable, distancia. Se entran los siguientes valores digitándolos en el espacio apropiado, pulsando luego Return o la flecha del cursor que señala hacia abajo.

INPUT	NAME	OUTPUT
(entrada)	(nombre)	(salida)
500	distancia	
8.5	tiempo	
	velocidad	
14	gasolina	
	gastosviaje	

Los valores de velocidad y gastosviaje se dejan en blanco para que los resuelva *TK!* Sus valores calculados se visualizarán en la columna OUTPUT. Para resolver velocidad y gastosviaje, pulse el signo de exclamación (!). Se visualizará la frase Direct Solver (solución directa) en la parte superior de la hoja Variable, porque al programa se le han proporcionado todos los datos necesarios para hallar una solución directa. Enseguida se visualizarán como salida los valores para velocidad y gastosviaje. Podemos eliminar los valores entrados previamente y obtener una cifra para distancia dándole a *TK!* nuevos valores para velocidad y tiempo, o para gastos del viaje y gasolina.

Hasta ahora los valores entrados en nuestro modelo no llevan asociada ninguna unidad. No podemos digitar millas ni galones en la columna de unidades de la hoja Variable, porque no se puede utilizar unidades hasta no haberlas definido. Desplazamos el cursor a la hoja Rule pulsando la tecla del punto y coma (;) y digitando luego =U. *TK!* reemplaza la hoja Rule de la ventana inferior por la hoja Unit. Ésta posee cuatro columnas:

From To	Multiply by	Add offset
(de) (a)	(multiplicar por)	(añadir resto)

El cursor se visualiza debajo de la palabra From. Ya

Conversión de unidades

podemos entrar las unidades que deseamos que *TK!* conozca y los valores de conversión.

Después pulse =R para recuperar la hoja Rule; seguido de ; para pasar a la hoja Variable. Ahora ya es posible entrar los nombres definidos en la columna Unit: m para distancia; h para tiempo; m/h para velocidad; g para gasolina y m/g para gastosviaje. Vacíe todos los valores en curso y sustitúyalos por éstos: 1,247 para distancia; 22.5 para tiempo

De sistema inglés a métrico

Ian McKinnell

y 43.9 para gastosviaje. Pulse ! para resolver y se visualizarán los valores métricos.

Ahora coloque el cursor sobre m, en la columna de unidades, y digite km de kilómetros. Pulse Return, y *TK!* convertirá automáticamente el valor de 1,247 para distancia de millas a kilómetros, de modo que el valor para distancia se cambia por 2006.423.

En este simple modelo hemos utilizado sólo algunas de las facilidades que ofrece *TK!*

En la línea del QL

Como respuesta a la aparición de máquinas más avanzadas, Sinclair ha vestido al Spectrum con nuevos ropajes

En la época de su lanzamiento, en la primavera de 1982, el Sinclair Spectrum ofrecía una notable relación calidad/prestaciones. Sus únicos competidores auténticos fueron el Vic-20, con unos escasos 3,5 K de memoria para el usuario, y el Texas TI99 4A, que se vendían al doble de su precio. De este modo el Spectrum se convirtió en un éxito instantáneo entre quienes adquirían un ordenador personal por primera vez, además de convertirse en la elección natural de los millares de entusiastas de los micros que desearon mejorar sus ZX80 y ZX81. La nueva máquina tenía una memoria asombrosa de 48 K, utilizaba un buen BASIC y ofrecía ocho colores para gráficos, así como una primitiva facilidad de sonido. El teclado, asimismo, representaba una gran mejora respecto a la lámina plástica plana "sensible al tacto" del ZX81. A la venta inicialmente sólo por correspondencia, el Spectrum fue un éxito inmediato y enseguida se convirtió en el micro más vendido.

En el tiempo transcurrido desde su lanzamiento, los competidores de Sinclair han producido una gama de máquinas para desafiar el dominio del mercado ejercido por el Spectrum. A pesar de tener un BASIC claramente inferior, el Commodore 64 fue el competidor que más éxito obtuvo; ofrecía más memoria (si bien necesitaba programas en lenguaje máquina para aprovecharla al máximo), un sonido excelente y un "verdadero" teclado, con teclas móviles tipo máquina de escribir. Asimismo, el BBC Micro ofrecía mejores especificaciones, pero su precio impidió que se convirtiera en una seria amenaza, y sus fabricantes optaron por no reducir su precio. Sin embargo, sucesivos recortes en los precios del Commodore redujeron sustancialmente el costo del 64. La reacción de Sinclair fue rebajar el precio del Spectrum.

Para entonces, el teclado del Spectrum (que en su tiempo fuera motivo de atracción) era ya un verdadero inconveniente. La base de software de la máquina no había sido superada y se produjeron muchos paquetes "serios" para ella. No obstante, tratar de utilizar programas para tratamiento de textos con el teclado del Spectrum era como escribir a máquina con un par de mitones puestos. Muchos usuarios, por consiguiente, invirtieron en teclados "adecuados" y esta tendencia se acentuó al aparecer la unidad Interface 1/Microdrive, tanto tiempo esperada. Enseguida se hizo evidente que los usuarios de micros de 1984 ya no estaban preparados para aceptar la idea de Sinclair de lo que constituía un dispositivo de entrada aceptable.

Seis en uno
Haciéndose eco de la moda actual de "empaquetar" software con los micros personales, Sinclair incluye con el Spectrum+ (y con el Spectrum de 48 K) un impresionante paquete de software compuesto por seis programas. Comprende un procesador de textos, una hoja electrónica, dos juegos y dos paquetes para gráficos

Cirugía plástica

La respuesta de Sinclair Research ha sido practicarle al Spectrum una cirugía plástica facial. El Spectrum+ es esencialmente la misma máquina, pero alojada en un teclado tipo QL, si bien reducido, con unas pocas teclas extras, un interruptor de Reset y un par de patas retráctiles. Todos los periféricos producidos para la versión más antigua trabajarán con el "Plus" (más), pero Sinclair no ha sabido aprovechar la oportunidad para colocar el rendimiento del Sinclair más en la línea de las máquinas rivales, mejorando el sonido o proporcionando un conector para monitor o una Interface 1 incorporada. Ahora las facilidades de sonido de la máquina constituyen su punto más débil. Las dos patas permiten que se escape un poco más de volumen de la base de la máquina, pero esto es más una incomodidad que una ventaja, puesto que significa que también se magnifican los ruidos propios de cargar (LOAD) y guardar (SAVE). El patético BEEP del Spectrum sigue siendo tristemente inadecuado.

El Spectrum+ mide 319 ×149 ×38 mm. El nuevo diseño hace más sencilla la programación al proporcionar teclas extras para las modalidades "Extended" y de gráficos, video normal e invertido, teclas Delete y Break y teclas separadas para los signos de puntuación que se utilizan comúnmente, como el punto y coma, las comillas, la coma y el punto. También se ha agregado una tecla Symbol Shift extra, y las teclas del cursor están ahora colocadas en lugares nuevos, a lo largo de una pequeña barra espaciadora. Todas las combinaciones de teclas antiguas siguen funcionando. A los antiguos usuarios del Spectrum el nuevo diseño podría causarles algunos problemas. En especial, la nueva tecla Edit está situada junto a la tecla "A"; si ésta se pulsa por error cuando se está entrando una línea de programa, la línea se pierde.

La placa de circuitos

En el interior de la atractiva carcasa tipo QL del Spectrum+ encontramos la placa de circuitos Issue 4.5. Esta es esencialmente la misma que la placa Issue 3 del Spectrum de 48 K que salió al mercado en agosto de 1983, con la adición de los desafortunados cables volantes del botón de reset parcheando la placa

Chris Stevens

SPECTRUM+

DIMENSIONES

319 x 149x 38 mm

MEMORIA E INTERFACES

Similares a las del Spectrum de 48 K, BASIC residente compatibilidad total de software

TECLADO

58 teclas esculpidas (incluyendo una barra espaciadora); teclado tipo membrana

DOCUMENTACION

Manual ilustrado en color con cassette de guía para el usuario

VENTAJAS

La riqueza del software existente para el Spectrum, los grupos de usuarios y las publicaciones especializadas constituyen envidiables atractivos

DESVENTAJAS

A pesar de las teclas nuevas y de las patas, el "tacto" y la acción del teclado continúan siendo un punto débil

Como parte del reempaquetamiento, los compradores del Sinclair reciben un lote compuesto por seis programas: *Psion chess, Make-a-chip, Scrabble, Chequered flag, Vu-3D* y el excelente software para tratamiento de textos *Tasword two*. Todos estos programas responden a un estándar muy elevado. Lamentablemente, no se puede decir lo mismo de la nueva documentación del Spectrum, la cual, a pesar de estar cuidadosamente presentada, carece de la profundidad del antiguo manual del Spectrum. Los editores sugieren, no obstante, que a medida que los usuarios adquieran mayor experiencia con la máquina, pueden hacerles llegar manuales más exhaustivos.

Para juegos, la nueva versión es ciertamente mejor que la original, pero los usuarios más entusiastas es posible que ya hayan invertido en una palanca de mando y una interface. Tanto la interface Kempston como la Fuller funcionan con el Spectrum+, si bien, como sucede con la máquina más antigua, el empleo de la Fuller Soundbox puede hacer inviable la ejecución de cierto software. La interface Centronics de Kempston también funciona a la perfección, al igual que el sistema Wafadrive de almacenamiento masivo.

El Spectrum+ representa, por cierto, una mejora sobre el Spectrum original, pero la idea que tiene Sinclair sobre lo que es un teclado razonable no va a contar con una aprobación universal. Si bien por parte de Sinclair se puede considerar una jugada inteligente la utilización de la tecnología del QL en un esfuerzo por hacer que el Spectrum resulte más atractivo a los compradores, hay que comparar el aumento en el precio del nuevo teclado con los accesorios ya disponibles. Bajo este prisma, no se puede considerar que la relación calidad/precio sea muy buena. Ciertamente, la nueva máquina tiene un aspecto más elegante, pero las teclas, a pesar de que están "esculpidas" para facilitar el tecleado, carecen de capacidad de respuesta y están demasiado apiñadas.

Para quien adquiera su primer ordenador, el Spectrum+ representa una opción que merece ser considerada, pero las facilidades que ofrece no compensan el aumento en su precio. Algunos usuarios podrían aventurar que Sinclair ha introducido el Spectrum+ simplemente como una forma de aumentar los precios; no sería sorprendente que el modelo original se retirara pronto de la venta. De hecho, la introducción de este modelo es un gesto extrañamente desinteresado por parte de Sinclair: seguramente habría sido más lógico rebajar el precio de la versión antigua (y del paquete Interface 1 Microdrive) y dejarlo así. Por otra parte, Sinclair podría haber aumentado ligeramente el precio e incluido todas las cosas que el Spectrum necesita *verdaderamente*, como unas adecuadas facilidades de sonido, un teclado de teclas móviles, conector para monitor e, incluso, un microdrive incorporado. Pero, de haber sido así, jamás hubiera estado listo a tiempo para las Navidades de 1984.

El teclado

Pensar que el teclado estilo QL del Spectrum+ representa una mejora o no, depende mucho del estilo de mecanografía que usted posea; la provisión de teclas separadas para las funciones más importantes sí representa una positiva innovación. Las secuencias de teclas originales del Spectrum (p. ej., [SYM SHIFT]+[0] para el punto y coma) duplican los efectos de las teclas nuevas

Ian McKinnell

Final del juego

Con la aparición de un fabuloso genio llega a su fin nuestro juego de aventuras en LOGO

Nuestro juego *El santuario de Zoltoth* sólo tiene incorporados dos "peligros". En CUARTO.4, el jugador se enfrenta a una serpiente poco amistosa, y el programa bifurca a un procedimiento de "peligro" especial:

```
TO ATAQUES.SERPIENTE
  PRINTL [[HAY UNA ENORME SERPIENTE] [QUE
  AVANZA LENTAMENTE HACIA TI!]]
END
```

El otro "peligro" no coloca al jugador en ninguna situación de peligro físico inmediato, pero, por cierto, podría causar problemas a largo plazo:

```
TO PUERTA
  PRINTL [[UNA GRAN PUERTA DORADA SE
  CIERRA DETRAS DE TI] [CERRANDOTE LA
  SALIDA POR EL SUR]]
  MAKE "PELIGROS []
  MAKE "LISTA.SALIDAS [[N 7] [E 8]]
END
```

Hay otras consideraciones que en ciertos puntos del programa es necesario tener en cuenta. El procedimiento COGER debe modificarse para que no se pueda recoger el anillo si se está llevando la espada.

```
TO COGERLO :ITEM
  IF :ITEM="ANILLO THEN COGER.ANILLO
  STOP
  AGREGAR.A.INV :ITEM
  SACAR.DEL.CUARTO :ITEM
END

TO COGER.ANILLO
  IF MEMBER? "ESPADA :INVENTARIO THEN PRINT
  [NO PUEDES RECOGER EL ANILLO] STOP
  AGREGAR.A.INV :ITEM
  SACAR.DEL.CUARTO :ITEM
END
```

Ésta es la única restricción en cuanto a que el jugador coja un objeto. Las siguientes rutinas permiten examinar lo que sea que esté sosteniendo.

```
TO EXAMINAR :OBJ
  IF :OBJ= "ANILLO THEN DESC.ANILLO
  STOP
  IF :OBJ= "COFRE THEN DESC.COFRE
  STOP
  IF :OBJ= "ESPADA THEN DESC.ESPADA
  STOP
  PRINT [NO LLEVAS NADA ESPECIAL]
END

TO DESC.ANILLO
  IF AQUI? "ANILLO THEN PRINTL [[EN EL ANILLO
  HAY UNA INSCRIPCION BORROSA:]
  [R-- --E]] ELSE PRINT [NO VEO NINGUN
  ANILLO]
END

TO AQUI? :OBJ
  IF MEMBER? :OBJ :CONTENIDO THEN OUTPUT
  "TRUE IF MEMBER? :OBJ :INVENTARIO THEN
  OUTPUT "TRUE OUTPUT "FALSE
END

TO DESC.COFRE
  PRINTL [[ES MUY HERMOSO] [Y
  EVIDENTEMENTE VALE UNA PEQUEÑA
  FORTUNA] [HAY UNA PEQUEÑA CALAVERA
  TALLADA] [EN UNA ESQUINA DE LA TAPA]]
END

TO DESC.ESPADA
  IF AQUI? "ESPADA THEN PRINT [ES DE ACERO]
  ELSE PRINT [NO VEO NINGUNA ESPADA]
END
```

El jugador necesita la espada para matar a la serpiente; si no la tiene, entonces es el ofidio el que lo mata a él.

```
TO MATAR :LA
  IF :LA="SERPIENTE THEN MATAR. SERPIENTE
  STOP
  PRINT [NO PUEDES HACER ESO!]
END

TO MATAR.SERPIENTE
  IF NOT MEMBER? "ATAQUES.SERPIENTE
  :PELIGROS THEN PRINT [NO VEO NINGUNA
  SERPIENTE] STOP
  IF MEMBER? "ESPADA :INVENTARIO THEN
  SERPIENTE.MUERE ELSE SERPIENTE.MATA
END

TO SERPIENTE.MUERE
  PRINT [LA SERPIENTE MUERE, ENROLLANDOSE
  EN SU AGONIA]
  MAKE "PELIGROS []
END

TO SERPIENTE.MATA
  PRINTL [[TU NO TIENES NINGUN ARMA] [CON LA
  CUAL MATARLA] [PERO AHORA YA ESTA
  TOTALMENTE ENLOQUECIDA] [TE MUERDE! TU
  ROSTRO SE ENSOMBRECE] [Y CAES AL SUELO
  RETORCIENDOTE]]
  MUERTO
END
```

El procedimiento MUERTO se anticipa sagazmente a aquellos jugadores que crean en la reencarnación. Si, después de haber muerto, usted digita cualquier otra cosa que no sea EMPEZAR, ¡el ordenador le recordará que está muerto!

```
TO MUERTO
  PRINT [ESTAS MUERTO!]
  PRINT1 "?
  MAKE "INPUT INSTRUCCION
  IF (:INPUT ="EMPEZAR)THEN EMPEZAR STOP
```

```
    PRINT [NO ME VENGAS CON ESO!]
    MUERTO
END

TO INSTRUCCION
    MAKE "INP REQUEST
    IF :INP=[] THEN PRINT1 "? OUTPUT
    INSTRUCCION
    OUTPUT FIRST :INP
END
```

Al frotar el anillo, ante los ojos del usuario hace su aparición un genio:

```
TO FROTAR :OBJ
    IF :OBJ= "ANILLO THEN FROTAR.ANILLO STOP
    PRINT [AHORA ESTA MUCHO MAS LIMPIO QUE
    ANTES]
END

TO FROTAR.ANILLO
    IF AQUI! "ANILLO THEN GENIO ELSE PRINT [NO
    VEO NINGUN ANILLO]
END
```

El genio le ofrece llevarlo a casa, pero si rechaza la invitación, entonces un fuerte viento lo llevará al azar hasta un cuarto situado en la parte oriental de la cueva:

```
TO GENIO
    PRINTL [[APARECE UN GENIO Y TE PREGUNTA:]
    ["QUIERES QUE TE LLEVE DE REGRESO A
    CASA?"]]
    PRINT1 "?
    MAKE "RESP FIRST INSTRUCCION
    IF ANYOF:RESP="SI :RESP="S THEN
    REGRESAR ELSE SOPLAR
END

TO REGRESAR
    PRINT [AL FIN EN CASA]
    IF MEMBER? "CETRO :INVENTARIO THEN PRINT
    [FELICITACIONES HAS ENCONTRADO EL CETRO!]
    ELSE PRINTL [[BUENO AL MENOS HAS
    ESCAPADO] [VIVO]]
END

TO SOPLAR
    PRINT [SOPLA UN FORTISIMO VIENTO]
    PRINT "
    MOVER1 (6+ RANDOM 5)
END
```

Mordedura mortal

Lo único que se puede abrir es el cofre, y éste contiene una araña venenosa. La calavera de la tapa es una advertencia para no abrirlo; pero, ya se sabe, ¡algunas personas nunca aprenden!, y las consecuencias, en este caso, son de imaginar.

```
TO ABRIR :OBJ
    IF :OBJ="COFRE THEN ABRIR. COFRE ELSE
    PRINT [NO LO PUEDES ABRIR]
END

TO ABRIR.COFRE
    PRINTL [[HAY UNA ARAÑA VENENOSA] [EN EL
    COFRE] [TE MUERDE]]
    MUERTO
END
```

Por último, he aquí un listado de todos los sustantivos del juego:

```
TO ESPADA
    OUTPUT "ESPADA
END

TO COFRE
    OUTPUT "COFRE
END

TO CETRO
    OUTPUT "CETRO
END

TO ANILLO
    OUPTUT "ANILLO
END

TO SERPIENTE
    OUTPUT "SERPIENTE
END
```

Si desea conservar el estado del juego para continuar en otro momento, simplemente digite SAVE "AVENTURA, y se guardará todo el contenido del espacio de trabajo. Se recuperará mediante READ "AVENTURA.

En muchos sentidos, el LOGO es un lenguaje ideal para programar juegos de aventuras. Existe, no obstante, un problema: en las implementaciones del lenguaje que existen hoy en día, no hay suficiente espacio. El juego que ofrecemos aquí apenas si cabe en la asignación de memoria para el Commodore 64. Cualquier ampliación que se introduzca en nuestro juego de aventuras en LOGO exigirá que el usuario adopte una decisión respecto a qué palabras conservar y cuáles suprimir.

Complementos al LOGO

Algunas versiones de LOGO MIT no poseen EMPTY?, ITEM, COUNT ni MEMBER? Las definiciones para las mismas ya las hemos ofrecido. En todas las versiones LCSI, utilice

```
EMPTYP por EMPTY?
LISTP por LIST?
MEMBERP por MEMBER?
TYPE por PRINT1
AND por ALLOF
OR por ANYOF
```

Hay una primitiva, EQUALP, que comprueba si sus dos entradas son la misma. Utilícela para comparar listas y palabras en lugar del signo de igualdad (que funciona para listas sólo en algunas versiones LCSI). La sintaxis IF en el LOGO LCSI se demuestra mediante:

```
IF EMPTYP :CONTENIDO[PRINT [NADA
ESPECIAL]] [PRINT :CONTENIDO]
```

Si la condición es verdadera se lleva a cabo la primera lista después de la condición; y, si es falsa, se lleva a cabo la segunda.
En el LOGO Atari utilice SE por SENTENCE, RL por REQUEST, y observe que ITEM no está implementada.
La versión que se ofrece en el texto se ejecutó en el Commodore 64; puede ser que otras máquinas no posean espacio suficiente para ejecutar todo el juego tal como está. Si fuera éste el caso, tendrá que reducir el tamaño del juego, omitiendo algunas palabras descriptivas.

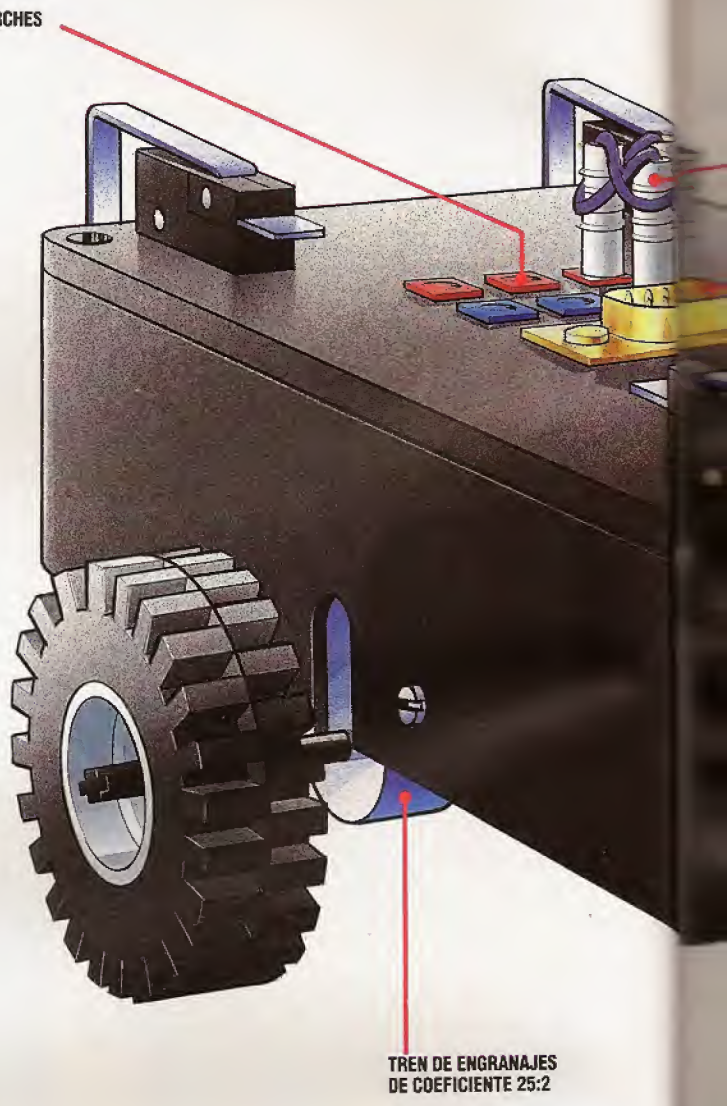
Liz Dixon

En el principio

Iniciamos un nuevo proyecto: la construcción de un robot con ruedas y sensores de proximidad y de luz

El robot estará alimentado por dos motores paso a paso, que accionarán dos ruedas mediante un sistema de engranaje. Los motores paso a paso que utilizaremos se pueden controlar para girar en pasos discretos de 7,5°. Accionar el motor a través de una caja de engranajes de un coeficiente de 25:2 significa que las ruedas del vehículo se podrán controlar con total precisión para una rotación del eje de 0,6°. Dado que los motores paso a paso trabajan girando a través de un ángulo discreto cada vez que reciben un impulso, resultan ideales para el control a través de un dispositivo digital. Emplearemos la puerta para el usuario del ordenador como nuestra fuente de control digital, lo que nos permitirá diseñar un software sencillo para usar conjuntamente con el robot. Además de estar equipado con motores paso a paso, el robot ya acabado contará con una gama de sensores, incluyendo sensores de proximidad y un par de sensores luminosos para permitir que siga una línea trazada en el suelo. Dado que se requieren cuatro líneas de datos de la puerta para el usuario para controlar los motores del vehículo, sólo quedan disponibles otras cuatro líneas para las entradas provenientes de los sensores. Para conseguir la máxima flexibilidad, el robot estará dotado de un sistema de "parches". Esto significa que a las cuatro líneas de datos disponibles se pueden conectar distintas combinaciones de sensores mediante algunos conectores montados en el robot y el empleo de cortos trozos de cable. Por ejemplo, puede que una aplicación requiera los cuatro sensores de proximidad, mientras que otra necesite dos sensores de proximidad y dos sensores luminosos. Con el sistema de parches los sensores requeridos se pueden enchufar directamente en las líneas de entrada de datos correspondientes.

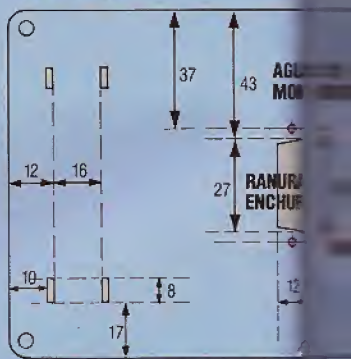
Al poder controlar el robot con toda precisión, y dado que el mismo estará dotado de sensores, asumiremos igualmente la tarea de diseñar algo de software refinado para posibilitar la creación de un mapa interno del entorno inmediato del robot. Entonces podremos comenzar a investigar los detalles propios de los algoritmos de planificación de ruta y estrategia de búsqueda. En este primer capítulo comenzamos con la construcción mecánica del robot. Ésta es bastante directa e implica perforar y cortar la caja plástica que forma la carcasa y el chasis del robot; el posicionamiento del tren de engranajes y de los agujeros para el montaje de los enchufes debe ser exacto.



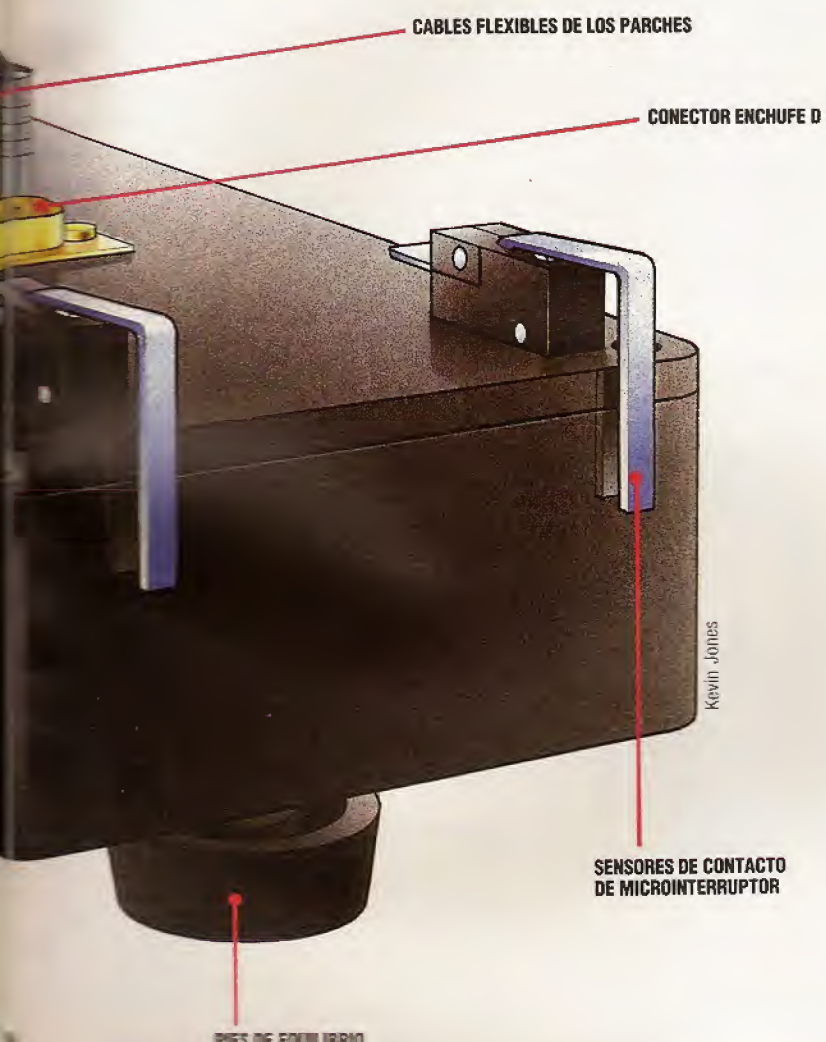
Primer paso

En primer lugar corte los agujeros necesarios en la carcasa plástica que ha de alojar y conformar el chasis del robot terminado. El diagrama ilustra la posición y el tamaño de los agujeros que se requieren. Los de los lados y la parte inferior de la caja van a recibir los ejes de las ruedas impulsoras. Los agujeros de montaje para el motor y el bloque de la caja de engranajes deben estar en línea a través de la caja. Los dos agujeros de la parte inferior van a recibir los dos pies que equilibran al robot sobre sus ruedas. El agujero de la tapa es para el conector tipo D en el cual se enchufará el cable conector del ordenador. Para cortar los agujeros grandes para la caja de engranajes y los ejes, quite la mayor parte del plástico con un cuchillo caliente o un soldador. Luego, con una pequeña lima, vaya formando el agujero pulcramente hasta alcanzar el tamaño adecuado

Tapa de la caja



TODAS LAS CIFRAS ESTAN EN MILIM



Lista de componentes

N.º	Artículo
1	40109
3	Conector DIL de 16 patillas
2	Resistencia de 100 ohmios
2	Resistencia de 0,5 watios 270 ohmios
2	Condensador de 0,1 µF
1	Condensador 25 V 1000 µF
1	Veroboard de 24 franjas × 50 agujeros
1	Bobina de cable estañado
1	Enchufe D 15 vías
1	Conector D 15 vías
1	Cubierta D 15 vías
1	Conector de potencia 2,1 mm
1	Conector IDC 20 vías
1	Conector marginal 24 vías
1	Caja 180 × 110 × 55 mm
1	Taco de tiras autoadhesivas
2	Pie de gabinete
1	Paquete tuercas 2BA
1	Paquete tornillos 6BA de 0,5 pulgadas
1	Paquete tuercas 6BA
1	Paquete tornillos M5 de 25 mm
1	Paquete tuercas M5

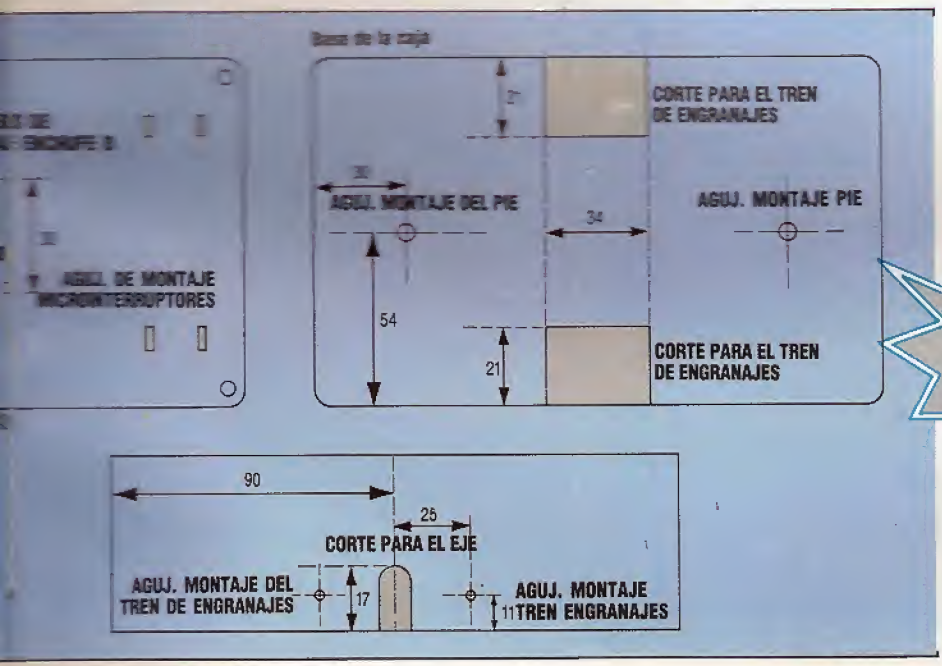
PIEZAS DE RADIO

2	Activador motor paso a paso SAA 1027
2	Motor paso a paso
2	Caja de engranajes sincrónica 25:2

VARIOS

2	Ruedas Lego de 62 mm
1	Paquete ejes technics
4 m	Cable plano de 12 vías
1 m	Cable plano de 20 vías
1	Fuente CD 12 V 1 amp
2	Tornillo 2BA × 4 cm

El importe total de estos componentes puede que sea un poco alto; por este motivo el robot será más atractivo como proyecto de grupo o escolar que como trabajo individual



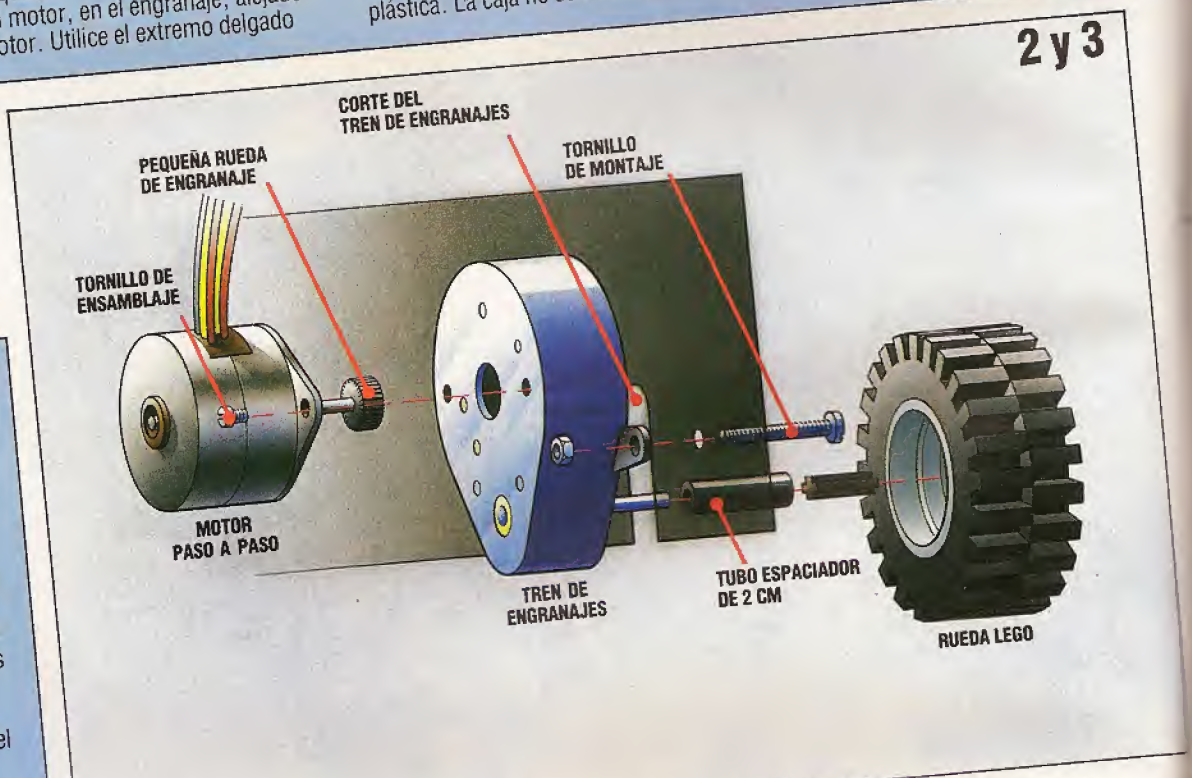
¡ATENCIÓN!

El robot consume una gran cantidad de energía; si la fuente de alimentación ha de activar asimismo la caja buffer, entonces el robot no tendrá potencia suficiente y no se moverá. Por consiguiente, el robot se debe conectar directamente a la puerta para el usuario de su ordenador. Siga fielmente las instrucciones: cualquier error podría dañar su máquina

Pasos 2 y 3

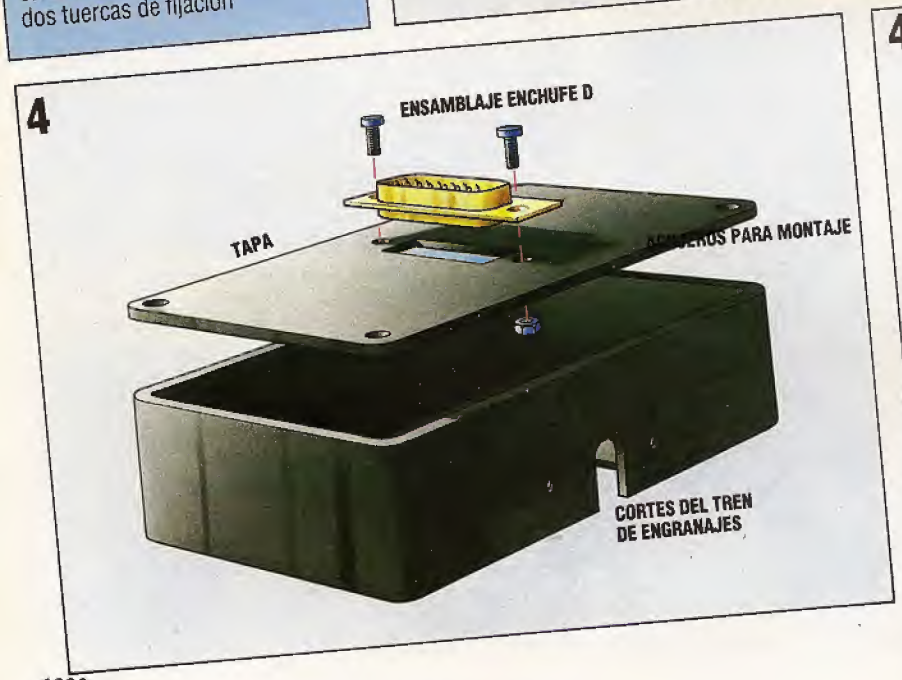
Los motores y la caja de engranajes se venden por separado y se deben ensamblar. La caja de engranajes viene acompañada por un pequeño engranaje de metal suelto y un pequeño tubo espaciador de plástico. El engranaje se debe encajar con la rueda dentada que sobresale del motor. Aplique un poco de pegamento Cyanoacrylate ("supercola") en el diámetro interior a través del engranaje y colóquelo, con el avellanador, en el eje del motor, en el engranaje, alejado del motor. Utilice el extremo delgado del tubo espaciador para alejar el engranaje correctamente del cuerpo del motor, tal como ilustra el diagrama. Deje transcurrir dos o tres horas para que la cola se seque. Monte el motor (y el engranaje) en la caja de engranajes con los cables del motor hacia el extremo más ancho de la caja. Con este objeto ésta trae dos tornillos. Tenga cuidado de encajar bien el engranaje con los engranajes internos de la caja cuando presione el motor en su sitio. Utilice los tornillos M5 para montar la caja en la carcasa plástica. La caja no se atornilla directamente en la carcasa, sino que se sostiene con el tornillo que se sujeta a ésta. El mismo hace que se puedan ajustar la caja y la rueda.

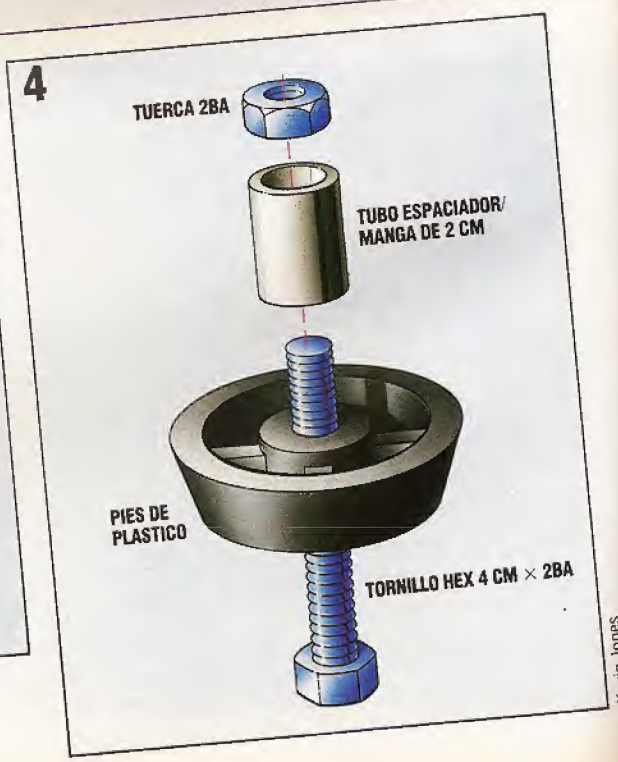
Las ruedas sólo van montadas en los ejes Lego especiales seccionados en x. Deslice unos 2 cm de la cubierta de plástico de un bolígrafo de diámetro adecuado sobre el husillo de la caja de engranajes como si fuera una manga. Asegúrela con supercola. Ahora pegue un eje Lego en el extremo de la manga. Utilice el más corto de estos ejes. Las ruedas se encajan a "presión" en los correspondientes ejes



Cuarto paso

Atornille el conector D (enchufe) macho en la tapa, con las patillas hacia arriba, utilizando los tornillos y tuercas 6BA. Monte los dos pies de equilibrio, empleando un tornillo 2BA a través de cada pie y ajustándolos en el interior de la carcasa con una tuerca. Los pies deben estar separados 3 cm de la base de los pies, medidos desde la parte inferior de la carcasa. Para ello coloque mangas de 2 cm entre el pie y la parte inferior de la carcasa. O bien se pueden utilizar dos tuercas de fijación





Kevin Jones

Puesta en escena

Ahora diseñaremos rutinas para describir los escenarios del juego y permitir que el jugador se desplace a través de ellos

Las descripciones básicas de todos los escenarios están retenidas en la matriz ES$ (véase p. 1247) y a ellas se puede acceder simplemente especificando el número del escenario al que se ha llegado. En *El bosque encantado*, la posición que ocupa el jugador en cualquier momento dado se almacena en la variable P y, por consiguiente, la descripción de ese emplazamiento está almacenada en ES$(P). Cuando se diseñaron originalmente los datos de los escenarios, se tuvo presente el contexto gramatical final de la descripción: ésta se redactó siempre de modo tal que pudiera ir precedida por "Ud. se halla...". Para un escenario dado, P, la descripción se puede formatear y visualizar mediante la utilidad desarrollada en el capítulo anterior, combinando "Ud. se halla" con la descripción retenida para ese emplazamiento en la matriz ES$(). Esto se puede apreciar en la línea 2010 del listado de *El bosque encantado*.

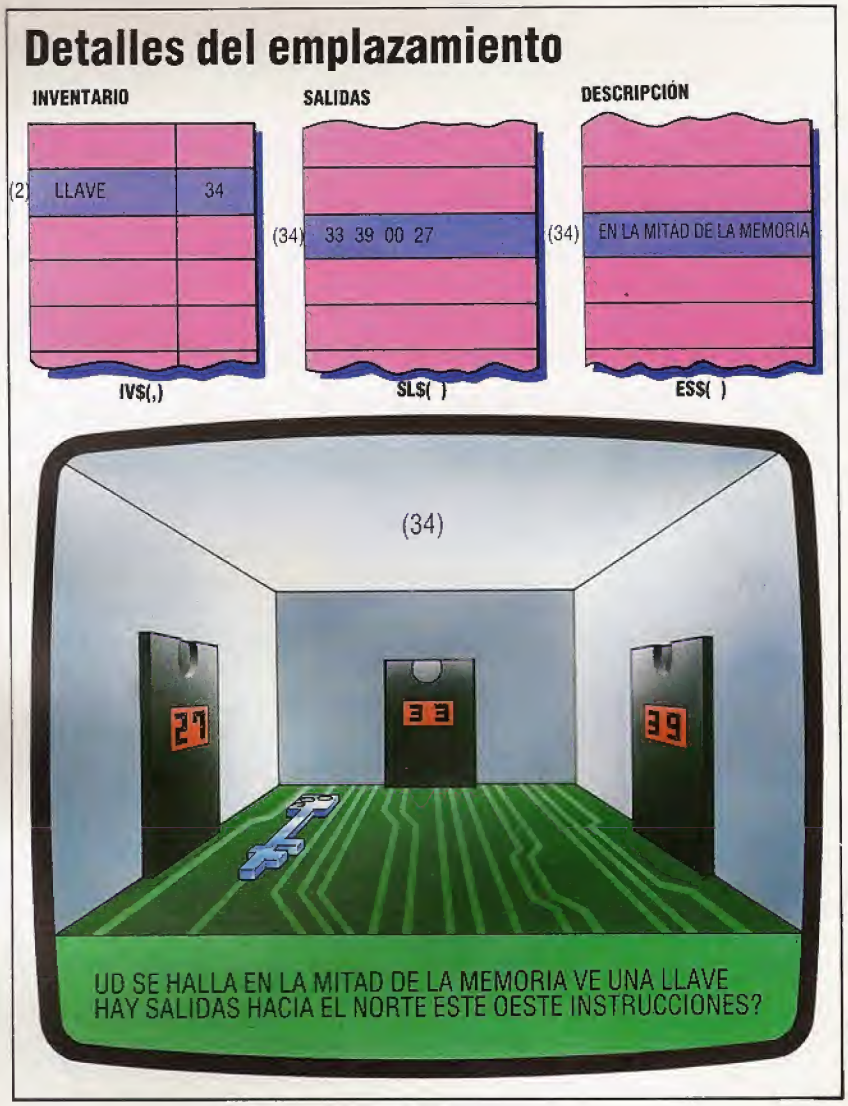
Además de la descripción básica del lugar al cual ha llegado, el jugador deseará saber también si ahí hay algún objeto. Los objetos utilizados en el juego están almacenados (junto con sus posiciones iniciales en el inventario) en una matriz bidimensional, IV$(,). Por ejemplo, IV$(N,1) contiene la descripción del objeto N del inventario, y IV$(N,2) contiene su posición. Si queremos determinar si en un emplazamiento determinado hay o no un objeto, debemos buscar en el inventario, cotejando la posición de cada objeto con el número del escenario que se está describiendo. Dado que en *El bosque encantado* sólo hay tres objetos, y ocho en *Digitaya*, se puede implementar una simple búsqueda lineal empleando un bucle FOR...NEXT.

El bucle de búsqueda utilizado en *El bosque encantado* está entre las líneas 2040-2080. Se explora la segunda columna de la matriz del inventario en busca de un emparejamiento con la posición en curso P. Cuando se encuentra una coincidencia, entonces se agrega la descripción correspondiente a la frase que describe los objetos. Puesto que en cualquier escenario en un momento dado podría haber más de un objeto, debemos dar lugar a la construcción de una frase en la cual se ofrezca una lista de objetos, separados cada uno por una coma. Mediante el empleo de SP$, inicialmente como una serie nula y luego como una coma, podemos insertar la puntuación correcta entre cada ítem. Un flag, F, establecido inicialmente en cero, se pone a uno para indicar el hecho de que en el transcurso de la búsqueda se ha encontrado un emparejamiento. Si al final de la búsqueda el flag continúa a cero, ello significa que no hay ningún objeto presente, y este hecho se le puede transmitir al jugador, como ocurre en la línea 2090 de *El bosque encantado*.

```
2000 REM **** DESCRIBIR ESCENARIO ****
2010 SN$="Ud. SE HALLA "+ES$(P):GOSUB5500
2020 SN$="VE"
2030 REM ** VERIFICAR INVENTARIO PARA OBJ **
2040 F=0:SP$=" "
2050 FOR I=1 TO 3
2060 IF VAL(IV$(I,2))<>P THEN 2080
2070 SN$=SN$+SP$+"UN "+IV$(I,1):F=1:SP$=", "
2080 NEXT I
2090 IF F=0 THEN SN$="NO "+SN$+"NINGUN OBJETO"
2100 GOSUB5500:REM FORMATEAR SALIDA
2110 RETURN
```

Los datos que contienen los detalles relativos a las posibles salidas desde cada escenario están retenidos en la matriz SL$(). Cada valor de la serie está compuesto por ocho dígitos. Subdividiendo estos ocho dígitos en grupos de dos, obtenemos (de izquierda a derecha) los números de los escenarios que hay hacia el norte, el este, el sur y el oeste del escenario en curso. Con el fin de determinar las posibles salidas, el programa primero divide la serie

Habitación con llave
Los detalles de los escenarios están almacenados en tres matrices en serie, que contienen nombres de objetos y su ubicación (V$), las salidas desde los escenarios (SL$) y descripciones (ES$). SL$(34), por ejemplo, podría contener el número de ocho dígitos 33390027, mostrando que el escenario 34 está conectado con los escenarios 33, 39 y 27. ES$(34) contiene "En la mitad de la memoria", que describe al escenario 34. IV$(2,2) contiene el número 34, indicando que IV$(2,1) (La llave) está en el escenario 34.



Kevin Jones

de ocho dígitos en los cuatro números que describen qué escenario hay en cada dirección.

```
2300 REM **** S/R DESCRIBIR SALIDAS ****
2310 SL$=SL$(P)
2320 NR=VAL(LEFT$(SL$,2))
2330 ES=VAL(MID$(SL$,3,2))
2340 SU=VAL(MID$(SL$,5,2))
2350 OE=VAL(RIGHT$(SL$,2))
```

Si en una dirección dada no hay salida, el valor asignado es cero: y esto es de gran ayuda para la descripción de las salidas. Se debe efectuar una verificación preliminar para ver si hay alguna salida posible antes de empezar a construir la oración "Hay salidas hacia el...". Esto se puede hacer llevando a cabo un OR lógico sobre las cuatro variables de dirección, y esto sólo producirá un resultado de cero si las cuatro variables de dirección fueran cero. Si no resultara de esta forma, entonces la rutina continúa para comprobar cada una de las variables de dirección de una en una. Si la variable no es cero, se le añade a la frase la dirección correspondiente.

```
2355 IF(NR OR ES OR SU OR OE)=0 THEN RETURN
2360 PRINT:SN$="HAY SALIDAS POR EL "
2370 IF NR<>0 THEN SN$=SN$+"NORTE "
2380 IF ES<>0 THEN SN$=SN$+"ESTE "
2390 IF SU<>0 THEN SN$=SN$+"SUR "
2400 IF OE<>0 THEN SN$=SN$+"OESTE "
2410 GOSUB 5500:REM FORMATEAR
2415 PRINT
2420 RETURN
```

Desplazamientos

Ahora que ya hemos desarrollado rutinas que describen cada escenario, podemos desarrollar procedimientos que le permitan al jugador hacer cosas en el mundo que hemos creado. En un futuro capítulo del proyecto estudiaremos algoritmos más detallados que analizan instrucciones. De momento nos ocuparemos de las instrucciones de movimiento que puede impartir el jugador simplemente entrando una instrucción de una sola palabra, como "NORTE" o "SUR". Si una de tales instrucciones se le pasa a una subrutina de movimiento como la variable NN$, entonces la rutina de movimiento es la siguiente:

```
3500 REM **** S/R MOVIMIENTO ****
3510 MF=1:REM ESTABLECER FLAG MOVIMIENTO
3520 DR$=LEFT$(NN$,1)
3530 IF DR$<>"N"ANDDR$<>"E"ANDDR$<>"S"ANDDR$
     <>"O"THEN GOTO3590
3540 IF DR$="N"AND NR<>0 THEN P=NR:RETURN
3550 IF DR$="E"AND ES<>0 THEN P=ES:RETURN
3560 IF DR$="S"AND SU<>0 THEN P=SU:RETURN
3570 IF DR$="O"AND OE<>0 THEN P=OE:RETURN
3580 PRINT:PRINT"NO PUEDES ";IS$
3585 MF=0:RETURN
3590 REM ** NOMBRE NO DIRECCION **
3600 PRINT"QUE ES ";NN$;" ?"
3610 MF=0:RETURN
```

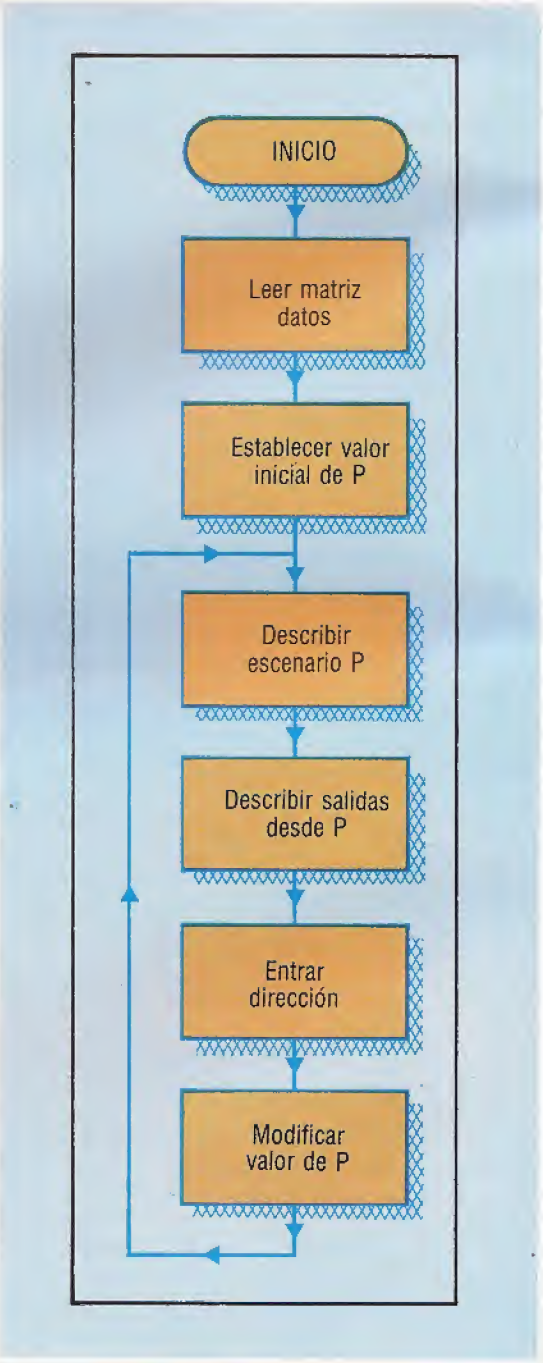
Esta rutina en realidad sólo emplea la primera letra de la instrucción de dirección que se le pasa. Comienza por comprobar que la instrucción sea realmente una dirección. De ser así, se actúa sobre la dirección especificada en la instrucción. Después de asegurarse de que hay una salida en esa dirección, se cambia P (la variable que lleva el registro de la posición del jugador) por el valor de NR, ES, SU u OE.

Sin embargo, antes de que podamos utilizar las subrutinas que hemos desarrollado aquí, necesitamos unirlas para que conformen un bucle repetitivo. El diagrama de flujo ilustra la estructura lógica de este bucle principal de llamada. Si bien ésta no es la estructura final del bucle principal del programa, sirve para demostrar los aspectos que hemos cubierto hasta ahora. Para utilizar las subrutinas dadas aquí, inserte las siguientes líneas, que constituyen una parte del bucle principal.

```
200 GOSUB6000:REM LEER DATOS MATRIZ
210 P=INT(RND(TI)*10+1):REM PUNTO DE PARTIDA
230 REM **** AQUI EMPIEZA EL BUCLE PRINCIPAL ****
240 MF=0:REM FLAG MOVIMIENTO
245 PRINT
250 GOSUB2000:REM DESCRIBIR POSICION
255 GOSUB2300:REM DESCRIBIR SALIDAS
260 PRINT:INPUT"INSTRUCCIONES";IS$
```

También incluya las siguientes líneas en el bucle de llamada principal:

```
270 NN$=IS$:GOSUB 3500:REM MOVIMIENTO
280 GOTO 230:REM RECOMENZAR BUCLE
    PRINCIPAL
```



Para el Spectrum

Debido a que el Spectrum retiene todas las matrices en serie como series de longitud fija, surgen problemas cuando deseamos imprimir uno de los elementos de una matriz en serie como parte de una frase más larga. Cuando se dimensiona una matriz en el Spectrum, el último número de la sentencia define la longitud de cada elemento de la matriz. Por ejemplo, DIM a$(3,2,20) dimensiona una matriz de tres por dos elementos, teniendo cada elemento una longitud fija de 20 caracteres. Si asignamos un elemento de la matriz a una serie de menos de 20 caracteres, entonces la diferencia se compensa agregando espacios al final de la serie. Esto supone un precioso espacio de memoria. Por lo tanto, debemos antes que nada eliminar cualquier espacio en cola. Para hacer esto, los usuarios del Spectrum deben digitar la siguiente rutina en el listado de *El bosque encantado*:

```
7000 REM **** PODADO SPECTRUM ****
7010 FOR I=LEN(A$)TO 1 STEP -1
7020 IF A$(I TO I)<>" " THEN LET N=I:LET I=1
7030 NEXT I
7040 LET S$=S$+A$(TO N)
7050 RETURN
```

Para el listado del *Digitaya*, digite estas mismas instrucciones pero emplee los números de línea del 8500 al 8550.

Esta rutina trunca A$, eliminando todos los espacios posteriores, antes de agregársela a S$. Recuerde que S$ es la variable utilizada para ensamblar una frase para su formateado. Para emplear esta rutina debemos pasar a la variable A$ el elemento de la matriz en serie (a incorporar a la oración), y después llamar a la subrutina. Por consiguiente, en las versiones de *El bosque encantado* y *Digitaya* para el Spectrum, hemos de introducir estos cambios:

El bosque encantado:

```
2010 LET S$="UD SE HALLA":A$=L$(P):
     GOSUB7000:GOSUB5500
2070 LET S$=S$+P$+"UN ":A$=V$(I,1):
     GOSUB7000:LET F=1:LET P$=", "
```

Digitaya:

```
1450 LET S$="UD SE HALLA":A$=L$(P):
     GOSUB8500:GOSUB5880
1500 IF VAL(V$(I,2))=P THEN LET
     S$=S$+P$+"UN ":A$=V$(I,1):
     GOSUB8500:LET F=1:LET P$=" "
```

Listado "Digitaya"

La estructura de *Digitaya* es similar a la de *El bosque encantado*. Agréguele las siguientes líneas a los listados que hemos ofrecido hasta ahora:

```
1100 GOSUB 6090:REM LEER DATOS MATRIZ
1210 PRINT:INPUT"INSTRUCCIONES";I$
1120 P=47:REM PUNTO DE PARTIDA
1130 :
1140 REM **** AQUI EMPIEZA EL BUCLE PRINCIPAL ****
1150 :
1160 MF=0:PRINT
1170 GOSUB1440:REM DESCRIBIR ESCENARIO
1180 GOSUB1560:REM LISTAR SALIDAS
```

Incluya, asimismo, estas líneas:

```
1220 NN$=I$:GOSUB 2000:REM MOVER
1230 GOTO 1140:REM REINICIAR BUCLE
     PRINCIPAL
```

Describir escenario y salidas

```
1440 REM **** S/R DESCRIBIR POSICION ****
1450 SN$="UD SE HALLA "+ES$(P):GOSUB5880
1460 SN$="VE "
1470 REM ** BUSQUEDA DE OBJETO **
1480 F=0:SP$=" "
1490 FOR I=1 TO 8
1500 IF VAL(IV$(I,2))=P THEN SN$=SN$+SP$+"UN"
     +IV$(I,1):F=1:SP$=" "
1510 NEXT I
1520 IF F=0 THEN SN$="NO "+SN$+"NINGUN OBJETO"
1530 GOSUB5880:REM FORMATEAR
1540 RETURN
1550 :
1560 REM **** S/R LISTA SALIDAS ****
1570 SL$=SL$(P)
1580 NR=VAL(LEFT$(SL$,2))
1590 ES=VAL(MID$(SL$,3,2))
1600 SU=VAL(MID$(SL$,5,2))
1610 OE=VAL(RIGHT$(SL$,2))
1620 IF(NR OR ES OR SU OR OE)=0THEN RETURN
1630 PRINT:SN$="HAY SALIDAS HACIA EL "
1640 IF NR<>0 THEN SN$=SN$+"NORTE "
1650 IF ES<>0 THEN SN$=SN$+"ESTE "
1660 IF SU<>0 THEN SN$=SN$+"SUR "
1670 IF OE<>0 THEN SN$=SN$+"OESTE "
1675 GOSUB 5880:REM FORMATEAR
1680 PRINT:RETURN
```

Subrutina Moverse a

```
2000 REM **** S/R MOVERSE A ****
2010 MF=1:REM ESTABLECER FLAG MOVIMIENTO
2020 DR$=LEFT$(NN$,1)
2030 IFDR$<>"N"ANDDR$<>"E" ANDDR$<>"S"ANDDR$<>"O"
     THEN2100
2040 IF DR$="N" AND NR<>0 THEN P=NR:RETURN
2050 IF DR$="S" AND SU<>0 THEN P=SU:RETURN
2060 IF DR$="E" AND ES<>0 THEN P=ES:RETURN
2070 IF DR$="O" AND OE<>0 THEN P=OE:RETURN
2080 PRINT"NO PUEDES ";I$
2090 MF=0:RETURN
2100 REM NOMBRE INCORRECTO
2110 PRINT"QUE ES ";NN$;" ?"
2120 MF=0:RETURN
```

Complementos al BASIC

Spectrum:

En los listados para ambos juegos, reemplace SL$() por E$(), SL$ por X$, SN$ por S$, I$ por T$, ES$() por L$(), NN$ por R$, SP$ por P$, DR$ por D$. En el listado de *Digitaya* sustituya las siguientes líneas:

```
1580 LET NR=VAL(X$(TO 2))
1590 LET ES=VAL(X$(3 TO 4))
1600 LET SU=VAL(X$(5 TO 6))
1610 LET OE=VAL(X$(7 TO))
2020 LET D$=R$(TO 1)
```

En el listado de *El bosque encantado*, sustituya las siguientes líneas:

```
2310 RANDOMISE:P=INT(RND(1)*10+1)
2320 LET NR=VAL(X$(TO 2))
2330 LET ES=VAL(X$(3 TO 4))
2340 LET SU=VAL(X$(5 TO 6))
2350 LET OE=VAL(X$(7 TO))
3520 LET D$=R$(TO 1)
```

BBC Micro:

En el listado de *El bosque encantado*, realice la siguiente sustitución de línea:

```
2310 P=RND(10)
```

Cangrejos

He aquí la versión para el Commodore 64 de este entretenido juego

Usted debe ayudar a un pobre camarón a regresar al mar, evitando a los voraces cangrejos que pueblan la playa. Cada camarón que logre adentrarse en el agua le proporcionará 10 puntos. Dispone de cinco rutas para intentar obtener su puntuación máxima. Utilice las teclas W para avanzar y Z para retroceder.

```
  5 REM ************
 10 REM * CANGREJOS *
 15 REM ************
 20 GOTO 1000
 50 PRINT N$;B$;
 60 PRINT N$;A$;
 70 PRINT BB$;BB$;
 80 PRINT N$;B$;
 90 PRINT N$;A$;
100 A$=RIGHT$(A$,1)+LEFT$(A$,39)
110 B$=RIGHT$(B$,39)+LEFT$(B$,1)
120 GET X$
130 P=P+40*((X$="W")-(X$="Z"))
140 IF P>PI THEN P=PI
150 IF P=PA THEN 2000
160 C=PEEK(P)
170 IF C<>32 AND C<>CP THEN 3000
180 POKE P1,CR
185 POKE P+N,9
190 POKE P,CP
200 P1=P
210 FOR I=1 TO 6
220 PRINT HH$;
230 NEXT I
240 T=T+1
250 IF T=500 THEN 3900
260 GOTO 50
1000 POKE 53280,6
1010 POKE 53281,7
1015 PRINT CHR$(147);
1020 X$=" "
1030 A$=" "
1040 B$=" "
1045 N$=CHR$(144)
1046 N=54272
1050 PA=1244
1060 PI=1604
1070 P=PI
1080 P1=P
1090 CP=81
1100 CR=32
1110 HH$=CHR$(145)
1120 BB$=CHR$(17)
1150 T$=CHR$(19)
1160 RESTORE
1165 FOR I=1 TO 40
1170 READ A
1180 A$=A$+CHR$(A)
1190 NEXT I
1200 B$=A$
1260 NP=5
1265 T=0
1270 S=0
1280 X=INT(RND(TI)*37)+2
1290 A$=RIGHT$(A$,X)+LEFT$(A$,40-X)
1300 PRINT T$;
1310 FOR I=1 TO 17
1320 PRINT BB$;
1330 NEXT I
1340 PRINT N$;"VIDA(S)[1SPC]REST.[1.SPC]:";NP
1350 PRINT
1360 PRINT N$;"PUNTOS[1SPC]:";S
1380 FOR I=1 TO 13
1390 PRINT HH$;
1400 NEXT I
1410 GOTO 50
2000 S=S+10
2010 POKE P1,CR
2020 P=PI
2030 P1=P
2040 GOTO 1280
3000 POKE P1,CR
3020 P=PI
3030 P1=P
3100 VO=54296
3110 WA=54276
3120 AA=54277
3130 HF=54273
3140 LF=54272
3150 SS=54278
3160 PH=54275
3170 PL=54274
3180 POKE VO,15
3190 POKE WA,65
3200 POKE AA,190
3210 POKE PH,15
3220 POKE PL,15
3300 FOR I=1 TO 11
3310 READ H
3320 READ L
3340 READ DL
3350 GOSUB 3700
3360 NEXT I
3370 RESTORE
3380 FOR I=1 TO 40
3390 READ A
3400 NEXT I
3500 NP=NP-1
3510 IF NP=0 THEN 4000
3520 GET X$
3530 GOTO 1280
3700 POKE HF,H
3710 POKE LF,L
3720 POKE SS,136
3800 FOR J=1 TO DL
3810 NEXT J
3820 POKE HF,0
3830 POKE LF,0
3840 RETURN
3900 PRINT CHR$(147);
3910 FOR I=1 TO 12
3920 PRINT
3925 NEXT I
3930 PRINT TAB(12)"**[1SPC]TIEMPO[1SPC] TRANSCURRIDO[1SPC]**"
3940 FOR I=1 TO 1000
3950 NEXT I
4000 IF S>R THEN R=S
4010 NB=0
4020 PRINT CHR$(147);
4030 FOR I=1 TO 7
4040 PRINT
4050 NEXT I
4060 PRINT TAB(16);"PUNTOS[1SPC]:";
4070 PRINT S
4080 FOR I=1 TO 4
4090 PRINT
4100 NEXT I
4110 PRINT TAB(12)"PUNTUACION[1SPC]MAXI MA[1SPC]:";
4120 PRINT R
4130 FOR I=1 TO 4
4140 PRINT
4150 GET X$
4160 NEXT I
4170 PRINT TAB(16);"OTRA[1SPC]?"
4180 GET X$
4190 IF X$=" " THEN 4180
4200 END
10000 DATA 117,184,105,32,32,117,184,105
10010 DATA 32,32,32,117,184,105,32,32
10020 DATA 117,184,105,32,32,32,32,32
10030 DATA 117,184,105,32,117,184,105
10040 DATA 32,32,117,184,105,32,32,32,32
20000 DATA 8,147,400,8,147,300
20010 DATA 8,147,100,8,147,400
20020 DATA 10,60,300,9,159,100
20030 DATA 9,159,300,8,147,100
20040 DATA 8,147,300,8,23,100
20050 DATA 8,147,400
```

Las tres últimas órdenes

Ya sólo nos quedan por diseñar tres órdenes para tener listo nuestro programa depurador, pero antes nos referiremos al mecanismo de interrupción

Antes de ocuparnos de las órdenes debemos tener en cuenta el mecanismo de interrupción, el cual es empleado en el programa original en los puntos de ruptura, donde hemos sustituido una instrucción con el opcode SWI (SoftWare Interrupt). El SWI, al igual que las restantes interrupciones del 6809, es accedido a través de una posición específica de memoria, a saber, la $FFFA. Esto significa que cuando se ejecuta un SWI los registros se guardan en la pila y el procesador carga la dirección de 16 bits que hay en $FFFA y $FFFB en el contador de programas (PC). Desde esa dirección comienza entonces la ejecución. Nuestra misión es cambiar este vector para que señale el punto de entrada de nuestro programa depurador. Aquí surge el problema de que los vectores suelen encontrarse en la ROM. El hecho de que estas direcciones sean fijas significa que el sistema operativo debe tener otros medios para "vectorizar" las interrupciones.

El sistema normal consiste en disponer de una tabla de salto (véase p. 1119) colocada en una zona de trabajo de la RAM, que es una parte de la memoria no disponible generalmente para programas y reservada a ciertos usos del sistema operativo. La dirección señalada en el vector contiene una instrucción JMP (*Jump*: saltar) seguida de una dirección, que por lo general volverá a señalar al sistema operativo. Pero esta dirección puede ser cambiada por la que se requiera, de modo que la primera instrucción ejecutada después de la interrupción de software será una JMP de salto a la dirección de entrada del programa depurador. Hay que cuidar de sustituir el contenido original de la tabla de salto antes de que nuestro programa acabe su ejecución, ya que siempre es posible que el sistema operativo ejecute después un SWI. Vale la pena recordar que el 6809 tiene tres interrupciones de software, y que no hay razón alguna para no usar SWI2 (opcode 10 3F y vector en $FFF4) o SWI3 (opcode 11 3F y vector en $FFF2), aunque el que éstas empleen opcodes de dos bytes hace necesario algunos cambios en el programa depurador.

Otro problema es que nuestro programa sólo puede ocupar cualquier sitio de la memoria que el programa a depurar deje libre. El depurador debe, pues, ser reubicable. De hecho habrá notado que todas las referencias a posiciones de memoria en el programa hecho o por hacer emplean un direccionamiento relativo al contador de programas. Pero en un cierto punto debemos saber la dirección absoluta del punto de entrada del programa para poder colocarlo en la tabla de salto de interrupciones. Tal dirección se tendrá que calcular en tiempo de ejecución, dado que el ensamblador no lo puede hacer.

Nuestra primera tarea será calcular esta dirección e insertarla en la tabla de salto. Se observará que la dirección del punto de entrada para el SWI será distinta de la dirección de inicio del programa depurador, puesto que la rutina existente en la dirección de inicio del programa debe encargarse del procedimiento de inicialización, que no se necesitará cuando se vuelva a entrar en el programa a través del SWI. Según esto, debemos ocuparnos de toda la inicialización dentro de una subrutina; el punto de entrada será la dirección que contiene la instrucción después de la llamada BSR a esta subrutina. Es muy conveniente que esta dirección sea precisamente la que se guardó en la pila por medio de la llamada BSR para que podamos tomarla de la pila y colocarla en el punto adecuado dentro de la tabla de salto.

Otra misión del procedimiento de inicialización es la de obtener la dirección de inicio del programa a depurar.

He aquí su diseño completo:

Procedimiento de inicialización

Data:
Vector-Dirección es la dirección que ha de encontrarse en $FFFA en X
Opcode-JMP es el opcode de la instrucción JMP en A
Dirección-Entrada es la dirección del punto de entrada en Y
Dirección-Inicio del programa a depurar en D
Proceso:
Tomar Vector-Dirección
Almacenar Opcode-JMP en Vector-Dirección
Tomar Dirección-Entrada
Almacenarla en (Vector-Dirección+1)
Tomar Dirección-Inicio del teclado
Guardarla

Ahora ya podemos afrontar la tarea de codificar las tres órdenes restantes. Algo que también debemos tener en cuenta afecta a una de estas órdenes, en concreto la orden R que visualiza los registros. Naturalmente no deseamos visualizar el contenido actual de los registros mientras se ejecuta el programa depurador, sino que queremos inspeccionar su contenido en el momento de ocurrir el punto de ruptura. Quiere decir que lo que nos interesa es saber los valores que la instrucción SWI hizo colocar en la

B	Insertar punto ruptura
U	Desinsertar punto ruptura
D	Visualizar puntos actuales de ruptura
S	Inicio ejecución programa
G	Ir o reanudar el programa desde donde se cortó
R	Visualizar el contenido de los registros
M	Inspeccionar y cambiar la posición de la memoria
Q	Abandonar el programa

pila. Sin embargo, puede haber sucedido que en la pila se hayan colocado encima otros valores cuando vayamos a inspeccionar la pila. Podemos calcular el número de bytes colocados en la pila que no nos interesan y descartarlo para obtener los valores de los registros. Pero es más sencillo guardar el valor del puntero de la pila en el momento de la interrupción y emplearlo después como referencia.

En la codificación de R supondremos que hemos realizado lo anterior y que no hay problema en obtener este contenido de los registros. La estructuración de la rutina es inmediata, sólo tenemos que tomar cada valor, uno tras otro, sin sacarlos de la pila y visualizarlos con etiquetas adecuadas. La única excepción será el valor S, que será el valor anterior a la interrupción y puede obtenerse añadiendo la cantidad precisa al valor guardado de S que empleamos como referencia de los valores de los registros colocados en la pila.

La orden R

Data:

Puntero-Pila es el indicador de la posición superior de la pila después de la interrupción en X
Valor-Byte-Individual retiene los valores de los registros de un solo byte en B
Valor-Byte-Doble retiene los valores de los registros de dos bytes en D
Etiquetas contiene las etiquetas relativas a los 9 regs.

Proceso:

```
Tomar Puntero-Pila
Cargar CC en Valor-Byte-Individual
Visualizar etiquetas(1), Valor-Byte-Individual
Repetir lo anterior para A, B y DP
Cargar X en Valor-Byte-Doble
Visualizar Etiquetas(5), Valor-Byte-Doble
Repetir lo anterior para Y, U y PC
Sumar 12 al valor original de Puntero-Pila
Visualizar Etiquetas(9), Puntero-Pila
```

Faltan dos órdenes más: la orden Q, abandonar (*quit*) el programa, que no necesita en sí una rutina especial; y la orden G, reanudar la ejecución tras el punto de ruptura. En este punto debemos sustituir la instrucción SWI que motivó la ruptura con la instrucción original reemplazada por ella y posteriormente devolver el control a dicha instrucción. Es fácil restablecer los registros a su contenido original con el solo empleo de RTI, que los hace salir a todos de la pila. Pero se debe tener en cuidado de que el valor del PC tomado de la pila sea el valor de la instrucción siguiente. Ya que el que deseamos se obtiene añadiendo un uno, debemos ajustar el valor en la misma pila antes de volver.

La orden G

Data:

Tabla-Puntos-Ruptura es tabla de 16 bits de tales pts.
Valores-Sustituidos es una tabla de opcodes sustituidos por instrucciones SWI
Punto-Ruptura-Siguiente es un número entre 1 y 16
Puntero-Pila es el valor guardado del puntero de la pila después de SWI

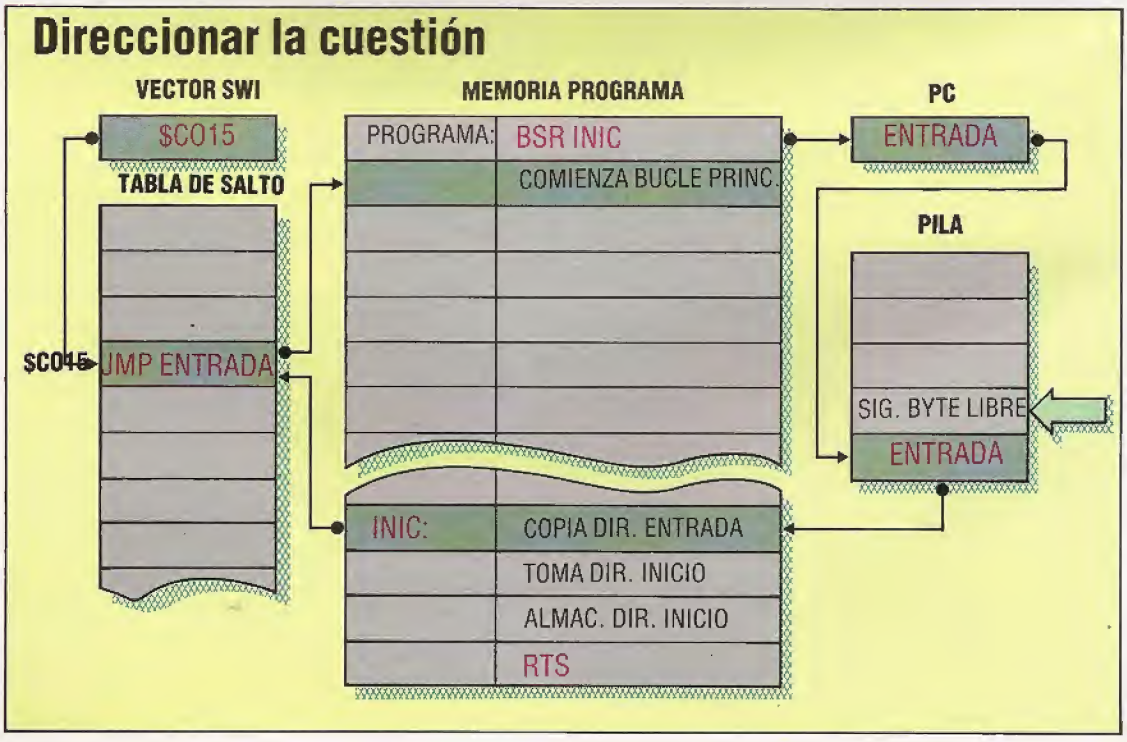
Proceso:

```
IF Punto-Ruptura-Siguiente >0 AND <=16 THEN
    Tomar opcode de Valores-Sustituidos (Punto-
    Ruptura-Siguiente)
    Almacenarlo en la dirección contenida en Tabla-
    Puntos-Ruptura (Punto-Ruptura-Siguiente)
    Colocar en S el valor de Puntero-Pila
    Decrementar el valor de PC en la pila
    Incrementar Punto-Ruptura-Siguiente
    Retorno de interrupción
ELSE
    Retorno de subrutina
```

Con la siguiente lección finalizaremos el curso de lenguaje máquina del 6809. En ella codificaremos el módulo principal de nuestro programa depurador y explicaremos cómo opera el programa.

A su debido tiempo
El programa depurador comienza con una llamada BSR a la rutina de inicialización, seguida del comienzo del bucle del programa principal. Una de las tareas de la inicialización es la de determinar la dirección absoluta de este comienzo de bucle y copiarla en la tabla de salto de interrupción, de modo que cuando se ejecute un SWI se pase el control por la tabla de salto y vuelva al comienzo del bucle. Esta dirección no puede conocerse por anticipado dado que el programa ha de ser totalmente reubicable. Por suerte, la dirección de retorno puesta en la pila por BSR es precisamente la dirección en cuestión, lo que quiere decir que la rutina de inicialización sólo necesita copiarla de la pila a la tabla de salto

Direccionar la cuestión



Procedimiento de inicialización

```
START   RMB   2            Para guardar dir. inicio (start)
OPJMP   FCB   $0E          Opcode-JMP
INIT    LDX   $FFFA        Toma Vector-Dirección
        LDA   OPJMP,PCR    Toma Opcode-JMP y lo guarda
        STA   ,X+          en Vector-Dirección
        LDY   1,S          Toma Dir.-Entrada de la pila
        STY   ,X           La guarda en Vector-Dir.+1
        BSR   GETADD       Toma del teclado Dir.-Inicio
        STD   START,PCR    La guarda
        RTS                Return
```

Orden R

```
STACKP  RMB   2            Puntero-Pila
LABELS  FCC   'CC A BDP X Y
                 UPC S'
SPACE   FCB   32           Espacio en código ASCII
CMDR    PSHS  A,B,X,Y      Guarda registros empleados
        LDX   STACKP,PCR   Toma Puntero-Pila
        LEAY  LABELS,PCR   Emplea Y para apuntar a la
                           etiqueta
        LDA   #4           Núm. de regs. de un solo byte
FOR01   BSR   CMDR1        Visualiza registro siguiente
        DECA               cuatro veces
        BGT   FOR01
        LDA   #4           Núm. de regs. de dos bytes
FOR02   BSR   CMDR2        Visualiza registro siguiente
        DECA               cuatro veces
        BGT   FOR02
        LDA   ,Y+          Primer carácter de la etiq.
        BSR   OUTCH        Lo visualiza
        LDA   ,Y+          Segundo carácter de la etiq.
        BSR   OUTCH        Lo visualiza
        LDA   SPACE,PCR    Visualiza el espacio
        BSR   OUTCH
        TFR   X,D          X contiene ahora el valor
                           requerido de S
        BSR   DSPADD       Visualiza S
        PULS  A,B,X,Y,PC   Restaura y retorna
* Subrutina para visualizar un registro de un solo byte
CMDR1   PSHS  A            Guarda A
        LDA   ,Y+          Primer carácter de la etiq.
        BSR   OUTCH        Lo visualiza
        LDA   ,Y+          Segundo carácter de la etiq.
        BSR   OUTCH        Lo visualiza
        LDA   SPACE,PCR    Visualiza el espacio
        BSR   OUTCH
        LDB   ,X+          Toma reg. sig. poniéndolo
                           en Valor-Byte-Individual
        BSR   DSPVAL       Visualiza Valor-Byte-Indiv.
        PULS  A,PC         Restaura A y retorna
* Subrutina para visualizar un registro de dos bytes
CMDR2   PSHS  A            Guarda A
        LDA   ,Y+          Primer carácter de la etiq.
        BSR   OUTCH        Lo visualiza
        LDA   ,Y+          Segundo carácter de la etiq.
        BSR   OUTCH        Lo visualiza
        LDA   SPACE,P-CR   Visualiza el espacio
        BSR   OUTCH
        LDD   ,X++         Toma registro siguiente y lo
                           pone en Valor-Doble-Byte
        BSR   DSPADD       Visualiza Valor-Doble-Byte
        PULS  A,PC         Restaura A y retorna
```

Orden G

```
BPTAB   RMB   32           Tabla-Puntos-Ruptura
REMTAB  RMB   16           Valores-Sustituidos
NEXTBP  RMB   1            Punto-Ruptura-Siguiente
CMDG    PSHS  A            Guarda A por si realizamos un
                           retorno normal
        LDA   NEXTBP,PCR   Punto-Ruptura-Siguiente
IF04    BLE   ENDF04       Si Punto-Ruptura-Siguiente>0
        CMPA  MAXBP,PCR    y <=16
        BGT   ENDF04       (núm. máx. de puntos ruptura)
        DECA               Conv. en desplaz. para tabla
        LEAX  BPTAB,PCR    Dir. de Tabla-Puntos-Rupt.
        LEAY  REMTAB,PCR   Dir. de Valores-Sustit.
        LDB   A,Y          Toma Valor-Sustituido
        LSLA               Convierte A en desplazamiento
                           para tabla de 16 bits
        STB   [A,X]        Lo almacena en la dirección
                           dentro de Tabla-Puntos-Ruptura
        LDS   STACKP,PCR   Coloca Puntero-Pila en S
        DEC   10,S         Ajusta valor del PC en pila
        INC   NEXTBP,PCR   Incrementa Punto-Rupt.-Sig.
        RTI                Return desde la interrupción
ENDF04  PULS  A,PC         Restaura y retorna
```

Estrellas en pantalla

"Starfinder", escrito para el BBC Micro, es un paquete educativo diseñado especialmente para astrónomos "amateurs"

Los astrónomos siempre se han encontrado con un problema en particular: el del tiempo atmosférico. Para los astrónomos profesionales es un hecho común pasarse semanas preparándose para un acontecimiento particularmente asombroso, sólo para descubrir que una densa nube oscurece la visión. Éste es el motivo por el cual en la actualidad la mayoría de los observatorios se construyen a gran altitud, en lugares donde exista poca nubosidad, o incluso en el espacio.

Ahora Century Software ha traído el universo hasta la pequeña pantalla a través de *Starfinder*, un paquete que contiene software en cassette y un libro. En esencia, el programa le permite al usuario contemplar cualquier sección del cielo de cualquier lugar del mundo en cualquier momento del s. xx.

Vista horizontal

Vista vertical

Luz de estrella, brillo de estrella
Las dos modalidades de visualización de *Starfinder* muestran un panorama del cielo rectangular o circular, según la línea de vista sea horizontal o vertical. La escena estelar se puede regular para un punto de vista situado en cualquier lugar de la superficie de la Tierra, en cualquier momento durante el s. xx. En la visualización se pueden buscar estrellas o planetas determinados, como el cometa Halley

Una vez que se ha cargado el programa desde la cassette, se le ofrece al usuario una lista de opciones. Una de ellas es observar el cielo en ese momento; la vista por defecto es la del cielo tal como se veía mirando hacia el sur desde Londres en la medianoche (hora del meridiano de Greenwich) del 21 de noviembre de 1984. Esto parece ser puramente arbitrario, ya que el panorama que ofrecía el cielo en esa fecha no tenía nada de especial: es probable que esto se haya diseñado así para coincidir con la fecha del lanzamiento del paquete y no con ningún acontecimiento cósmico específico.

En la parte superior de la pantalla el programa visualiza la hora y la fecha, y da la situación del observador en términos de longitud y latitud. Debajo de esta información está la carta estelar propiamente dicha, que muestra el cielo nocturno de suroeste a sureste, con una altitud (el ángulo de vista) de 60°. Al programa le lleva algunos segundos visualizar las estrellas, puesto que para procesar y trazar cada uno de los cuerpos celestes se requiere gran cantidad de cálculos numéricos.

Las estrellas se representan como cuadrados blancos (el programa se ejecuta en la modalidad 4) y las estrellas más brillantes se indican con un mayor tamaño. Es una verdadera lástima que las máquinas Acorn no posean una instrucción de "brillo", lo que haría más realista la visualización. Los planetas también tienen forma cuadrada, pero están trazados en rojo, mientras que el Sol se representa mediante un gran cuadrado amarillo y la Luna por medio de un pequeño punto amarillo. Utilizando la "sonda espacial", que se mueve pulsando las teclas para el cursor, se puede identificar cualquiera de las estrellas reflejadas. Cuando se coloca la sonda espacial (una cruz roja) sobre una estrella, aparece arriba del mapa el nombre científico y el nombre corriente de esa estrella, así como sus coordenadas, dadas como una cifra de altitud y de acimut (la altitud se expresa como una cifra positiva o negativa, con el horizonte como cero; el acimut indica los grados este u oeste proa al norte). El panorama se puede cambiar desde el teclado modificando cualquiera de estas cifras.

Regresando al menú principal, el usuario puede modificar la vista para que corresponda a la que se observó en cualquier momento en cualquier lugar del mundo. El programa también se puede utilizar para encontrar un cuerpo celeste determinado: una estrella, un planeta o el cometa Halley.

A pesar de que este paquete es muy amplio, tiene sus limitaciones. La principal de ellas es la cantidad de estrellas que se pueden visualizar. El programador, Ronald Alpiar, ha optado por incluir estrellas de magnitud 4 o inferior (la "magnitud" es el nivel de brillo de una estrella, representando una magnitud alta a una estrella débil). A simple vista el ojo humano puede distinguir cuerpos celestes de magnitud 6, de modo que el programa no intenta mostrar todas las estrellas que podría ver una persona en una noche clara.

El libro que se incluye con el programa analiza los diferentes tipos de telescopios e informa de las mejores horas para observar los astros.

Starfinder: Para el BBC Micro y el Electron
Editado por: Century Software, Portland House, 12-13 Greek Street, London, W1V 5LE, Gran Bretaña
Autores: Heather Couper (el libro), Ronald Alpiar (el programa)
Palanca de mando: No se necesita
Formato: Cassette

El nombre del robot

Examinemos algunos de los productos que se venden comercialmente bajo la denominación de robot

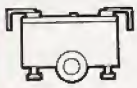
Hasta ahora en nuestra serie sobre Robótica nos hemos ocupado fundamentalmente de consideraciones teóricas relativas al diseño y el funcionamiento de los robots. En la práctica, no se han implementado muchos de los conceptos que hemos analizado, o se los ha limitado debido a una cierta falta de componentes mecánicos y/o de software inteligente. Los robots actuales, ya sea los destinados al uso doméstico o al uso industrial, tienden a quedarse cortos en relación a lo que hemos llegado a esperar de ellos con el paso de los años. Existen sensores para dotar al robot de vista, oído y tacto, pero las "sensaciones" que éste experimenta no significan nada para él, y no se las puede sintetizar para estimularlo hacia un comportamiento original, no programado. Robbie el Robot y sus otros compañeros de ficción todavía están muy lejos de hacerse realidad.

No obstante, en la actualidad se están vendiendo muchos productos bajo la denominación común de *robot*. Éstos van desde pequeños y baratos juguetes hasta el sofisticado R2D2 y robots industriales sumamente costosos. Después de haber examinado los componentes del diseño de robots y los aspectos teóricos, debemos ahora considerar qué constituye un auténtico robot. No hemos de ser demasiado exigentes, pero estamos en condiciones de reunir todo cuanto sabemos y aplicar estos conocimientos para elaborar una definición funcional.

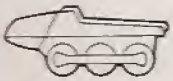
La primera consideración, que viene a eliminar muchos de los productos "robot" de precio más reducido, es el movimiento: ¿puede el robot desplazarse solo a través de un espacio? No podemos esperar que el robot se programe a sí mismo, ni que establezca un curso de acción sin la guía del hombre; pero sí podemos esperar que, una vez puesto en movimiento, sea capaz de operar con independencia del control humano continuo. Sin esta libertad de movimiento no se puede considerar que un objeto sea un robot.

Habiendo superado la prueba del movimiento, nuestro candidato a robot debe evaluarse sobre la base de cómo se efectúa la acción. A un pequeño coche de juguete se le puede instalar un motor y unas pilas que lo mantengan en movimiento siguiendo una línea recta. Agréguele parachoques y el coche podrá desviarse apartándose de obstáculos tales como paredes y mesas. Póngale al coche un centro de gravedad ligeramente inusual y grandes neumáticos de goma y podrá incluso ascender por

Tres en el laberinto



Robot
Alimentado y controlado desde su ordenador madre, el robot está equipado con sensores sensibles al tacto y a la luz



Big Trak
Este dispositivo activado por microprocesador se puede programar a través de su teclado con instrucciones de distancia y dirección similares a las del LOGO

Coche con parachoques
Este juguete a pilas avanzará en línea recta hasta chocar contra algún objeto, en cuyo caso su trayectoria variará sin rumbo fijo

Steve Cross

Pistas sorprendentes
Los tres dispositivos están intentando recorrer un laberinto: el coche de juguete simplemente avanza tropezando de pared en pared, el Big Trak sigue las instrucciones que ha programado su operador humano para superar el laberinto, mientras que nuestro robot se aprende el laberinto a través de la interacción de su software y sus sensores. Al final, el robot conseguirá resolver el laberinto, independientemente de lo que suceda; el Big Trak seguirá su programa, de modo que, en caso de que las directrices del operador sean correctas, podrá superarlo; el coche de juguete sólo podría resolver laberintos "diestros" y, aun así, sería por casualidad. Cuando el coche choca con el Big Trak, el hecho no lo afecta, puesto que su comportamiento no tiene ningún propósito determinado; el Big Trak, sin embargo, sufrirá una desviación de 90° en su recorrido (señalado en color verde) pero seguirá girando y avanzando como si aún estuviera en su camino (señalado en rojo). Ambos dispositivos no reaccionan de forma "inteligente" ante este acontecimiento imprevisto; ante esta situación, el robot se comportaría como si se tratara de un aspecto más de un entorno imprevisible

una pared y darse la vuelta, siguiendo luego en otra dirección. El coche se mueve por sí mismo y es independiente de todo control humano, pero ¿se puede decir que sea un auténtico robot? La respuesta, bastante evidente, es "No", pero para nuestro estudio sobre los robots es de fundamental importancia entender por qué esto es así.

El movimiento del robot, como ya hemos visto, se divide en dos categorías: movimiento simple bajo el control de un programa, y movimiento inteligente. En ambos casos la clave es la programación. Nuestro coche activado por motor no puede ser un robot porque no se lo puede programar para que se mueva de diversas formas. Posee un patrón de operaciones establecido incorporado en él mecánicamente, pero no puede responder a instrucciones humanas y no posee medio alguno en virtud del cual un controlador humano pueda generar un movimiento alternativo. Una interesante variación en este sentido es el coche, autocamión u otro objeto activados por motor que se pueda programar perforando un patrón en una ficha. La ficha se lee mecánicamente y el coche sigue el patrón girando hacia la izquierda o la derecha y siguiendo hacia adelante de acuerdo a las instrucciones de la ficha. Según nuestra definición, este tipo de coche sí sería un robot, porque puede ser programado con el software de ficha perforada. La necesidad de un movimiento programable elimina a muchos otros productos pequeños y baratos etiquetados bajo el denominador común de "robot".

Se pueden aplicar otras varias consideraciones. ¿Posee el objeto sensores que le den una entrada proveniente del mundo exterior? ¿Está capacitado para responder a su entorno y crear un modelo interno cambiante? ¿Puede jugar una partida de ajedrez? A nuestro robot se le podría aplicar cualquiera de estos criterios, pero, en el análisis final, el elemento del movimiento programable es esencial.

Los siguientes productos, todos ellos vendidos como robots, se pueden adquirir en los comercios especializados.

NOMBRE: Beasty
TIPO: Brazo-robot
PROGRAMABLE: Sí
PROVISTO DE SOFTWARE: Sí
REALIMENTACION SENSORIAL: Realimentación posicional
DISTRIBUIDORES: Commotion Ltd, 241 Green Street, Enfield, Middlesex EN3 7SJ, Gran Bretaña

Recibe sus instrucciones desde la puerta para el usuario del BBC Micro y se alimenta a partir de la fuente eléctrica auxiliar del ordenador. Se suministra en forma de modelo para armar e incluye tres servomotores. También se acompaña de dos manuales, un folleto de construcción y un manual de operaciones. El robot lleva incorporado su propio sistema operativo (Robol) que permite que el usuario mueva cada uno de los servomotores de forma independiente.

NOMBRE: Hebot II
TIPO: Robot móvil
PROGRAMABLE: Sí
PROVISTO DE SOFTWARE: Sí
REALIMENTACIÓN SENSORIAL: Táctil
DISTRIBUIDORES: Powertran Cybernetics Ltd, West Portway Industrial Estate, Andover, Hampshire, SP10 3NN, Gran Bretaña

Es una tortuga-robot accionada por dos motores CD conectados a dos ruedas. La tortuga se conecta en interface al conector marginal del Sinclair ZX81, si bien los fabricantes aseguran que, mediante la reconexión de algunos cables, se puede hacer que funcione en cualquier micro personal. Viene en forma de modelo para armar con un folleto de construcción. La tortuga posee un lápiz retráctil y cuatro detectores de colisión que proporcionan realimentación táctil. El suministro eléctrico le llega desde el ordenador.

NOMBRE: Memocon Crawler
TIPO: Robot móvil
PROGRAMABLE: Sí
PROVISTO DE SOFTWARE: No
REALIMENTACIÓN SENSORIAL: No
DISTRIBUIDORES: Prism Products, Prism House, 18-29 Mora Street, London, EC1, Gran Bretaña

Es el robot más sofisticado de la gama Prism Movits. Emplea pilas de 1,5 V para alimentar dos motores eléctricos CD. El control se efectúa mediante una caja que posee cinco teclas, una para cada una de las instrucciones. Esta caja está conectada al Crawler mediante un cable plano. El dispositivo posee, asimismo, un zumbador y LED (diodos emisores de luz); éstos también se pueden controlar desde el teclado.

NOMBRE: Valiant Turtle
TIPO: Robot móvil
PROGRAMABLE: Sí
PROVISTO DE SOFTWARE: Sí
REALIMENTACIÓN SENSORIAL: Posicional
DISTRIBUIDORES: Valiant Designs, 1st Floor, Park House, Battersea Park Road, London, SW11, Gran Bretaña

Tiene realmente forma de tortuga. Está activado mediante un par de motores; cada uno le proporciona potencia a una única rueda. El dispositivo se controla desde el ordenador mediante un haz infrarrojo. El software que se suministra con el dispositivo está diseñado para ejecutarse mediante LOGO, si bien funcionará igualmente sin el lenguaje de instrucciones. La potencia se la suministra una pila recargable incorporada en el aparato. Tiene un portalápiz que le permite trazar diseños mientras avanza.

NOMBRE: BBC Buggy
TIPO: Robot móvil
PROGRAMABLE: Sí
PROVISTO DE SOFTWARE: Sí
REALIMENTACIÓN SENSORIAL: Sensor luminoso, dispositivos de presión
DISTRIBUIDORES: Economatics Education Ltd, Epic House, 9 Orgreave Road, Handsworth, Sheffield, Gran Bretaña

Es una tortuga controlada mediante software. Viene en forma de modelo para armar y se opera desde la puerta para el usuario del BBC Micro, con potencia proveniente de la fuente de alimentación auxiliar del ordenador. Utiliza un par de motores paso a paso que le proporcionan un movimiento de gran precisión. Puede responder a una fuente luminosa, buscarla y encontrarla; se le pueden instalar un lápiz y un lector de código de barras.

NOMBRE: Edinburgh Turtle
TIPO: Robot móvil
PROGRAMABLE: Sí
PROVISTO DE SOFTWARE: Sí
REALIMENTACIÓN SENSORIAL: Posicional
DISTRIBUIDORES: Jessop Acoustics, Unit 5, 7 Long Street, London, E2, Gran Bretaña

A esta tortuga se le ha dado el nombre de Edinburgh en honor a la ciudad en la cual se desarrolló. La Edinburgh Turtle se conecta al ordenador mediante un cable plano, desde donde también obtiene su potencia. La activan un par de motores eléctricos CD; cada uno alimenta a una sola rueda. Está equipada con un portalápiz retráctil y un altavoz incorporado. La firma fabricante acaba de introducir una versión a control remoto.

Conjetura exacta

Concluiremos el estudio del "TK!Solver" con un detenido análisis de algunas de sus características exclusivas

Tal como explicábamos en el capítulo anterior, *TK!Solver* es un paquete de software de la "próxima generación" que introduce el concepto de hoja electrónica en el reino de las matemáticas y la ingeniería de nivel superior. Ya hemos visto que el programa permite que el usuario defina variables con nombres y utilice las mismas en ecuaciones matemáticas complejas. En este capítulo, el último de nuestra serie sobre hojas electrónicas, analizamos con todo detalle la inusual capacidad del *TK!* para *iterar*. Éste es un método en virtud del cual el programa puede resolver una variable haciendo suposiciones respecto a la misma. Normalmente, cuando se trabaja con ecuaciones, uno puede determinar los valores de todas las variables si desde el comienzo se cuenta con la información suficiente. El programa sencillamente reduce el programa a una serie de cálculos. Por ejemplo:

$$A^2+B^2=2COS\ Y$$

Cualquiera de estas tres variables se puede resolver fácilmente si se conocen los otros dos valores. Enfrentado a esta ecuación (y proporcionándole valores para A y B), el Direct Solver (solucionador directo) del *TK!* llevará a cabo los cálculos requeridos y producirá un valor para Y.

Pero existen ocasiones en las cuales la determinación de un valor no es directa. La ecuación redundante, que define a una variable en términos de sí misma, constituye uno de tales casos. Por ejemplo, tomemos:

$$D=(A+B)/(2*D)$$

en un modelo en el cual el único valor conocido es A. Se pueden plantear otros problemas a consecuencia de un modelo incompleto, o de un modelo con muchas variables independientes y una cantidad limitada de datos. El concepto de iteración es difícil, de modo que tomemos un ejemplo de iteración que sea más práctico.

Volvamos a considerar el modelo del viaje en coche que creamos en el capítulo anterior y agreguemos unos pocos detalles para hacerlo más aplicable. Como recordará, el modelo previo se construyó alrededor de cinco valores: distancia, tiempo, velocidad, gasolina y gastosviaje. Podría calcular los gastos del viaje, disponiendo de la velocidad y el consumo de gasolina; la distancia, a partir de la velocidad y el tiempo, y algunas otras variaciones sencillas. Pero ¿y si ahora quisiéramos determinar a qué velocidad deberíamos viajar para poder completar el viaje con un presupuesto dado?

En primer lugar, debemos añadir varios factores a nuestro modelo. Por ejemplo, el modelo debería tener en cuenta la potencia del vehículo, la fricción interna del motor y la resistencia del viento, todos los cuales producen un efecto en los gastos de viaje y la velocidad del vehículo. (Supondremos que la fricción interna es constante.) Asimismo, debemos fijar un límite máximo para nuestro presupuesto y el costo del combustible que se esté consumiendo.

Se comienza a construir el verdadero modelo entrando estas ecuaciones en la hoja Rule, una ecuación en cada línea. Estas ecuaciones se leen en la hoja Variable de forma automática. Al haber varias

Construcción de ecuaciones

variables, la pantalla queda demasido pequeña para contener toda la información. Para ver visualizadas todas las variables, podemos hacer que la hoja Variable aparezca sola en una ventana. Esto lo conseguimos pulsando la tecla del punto y coma (;) para trasladar al cursor hasta la ventana de variables, y digitando luego W1. Ahora se pueden ver todas las variables y podemos comenzar a entrar valores para ellas.

Solución directa

El modelo se puede resolver directamente si al principio se le suministra información suficiente.

Entrada de valores para el Direct Solver

Por ejemplo, si entra los siguientes valores en la columna INPUT: 5000 a DISTANCIA, 45 a VELOCIDAD, 1.5 a PRECIO, .75 a FRICCION y .0000085 a VIENTO, y después pulsa ! para llevar a cabo los cálculos, *TK!*

Resultados del Direct Solver

visualizará el mensaje Direct Solver, y los valores desconocidos aparecerán en la columna OUTPUT.

Ello nos proporciona un claro análisis de toda la información que deseamos. Pero, ¿y si deseáramos utilizar este modelo para resolver un problema habiendo dado desde el comienzo menos información? Tomemos un cálculo en el cual tenemos un presupuesto máximo de 50 libras para gastar en gasolina en un viaje de 1 000 millas. Sabemos el precio de la gasolina (1.75 libras el galón), de modo que podemos determinar cuánto podemos gastar por milla. Sería más difícil, sin embargo, determinar a qué velocidad debemos viajar para completar el recorrido sin pasarnos del presupuesto.

Empezamos por poner a cero los valores que ya habíamos entrado. Esto lo hacemos digitando RVY (de *Reset Variables Yes*: poner variables a cero sí). Luego entramos la información que ya conocemos: 1000 para distancia, 50 para costo, y 1.75 libras para precio. Utilizamos un valor de 1/3 para la fricción interna (que *TK!* redondea a 0.333333) y 0.0000095 para la resistencia del viento. Pulse !

Modelo incompleto

para calcular, y aparecerán los valores correspondientes. Podrá observar que no se genera ningún valor para velocidad, tiempo y potencia, y la velocidad es el valor específico que necesitamos. Si pasamos la visualización de la hoja Variable a la hoja Rule, veremos que tres de nuestras ecuaciones

Ecuaciones no satisfechas

están sin satisfacer (lo que se indica mediante el * de la columna Status).

Solucionador iterativo

Dado que no podemos resolver el modelo utilizando el Direct Solver, debemos probar con el Iterative Solver (solucionador iterativo). Éste toma un valor inicial, entrado a modo de conjetura o hipótesis, y lo acomoda en la ecuación. Si el valor no es correcto, el *TK!Solver* emplea una serie de aproximaciones sucesivas para determinar con precisión el valor exacto.

Lo primero que debemos hacer es tomar el valor de gastos de viaje generado previamente y trasladarlo a la columna INPUT para suministrarle a *TK!* un valor extra con el cual comenzar. Esto lo hacemos digitando I en la columna Status junto a gastos de viaje en la hoja Variable. Luego estimamos un valor para velocidad (50, p. ej.); entre este número en la columna Input, digite G de *guess* (conjetura) en la columna Status y pulse ! para calcular. *TK!*

Valores iterados

visualiza Iterative Solver en la parte superior de la pantalla y un contador de cada aproximación. Al cuarto intento, *TK!* obtiene el valor correcto para velocidad, tiempo y potencia.

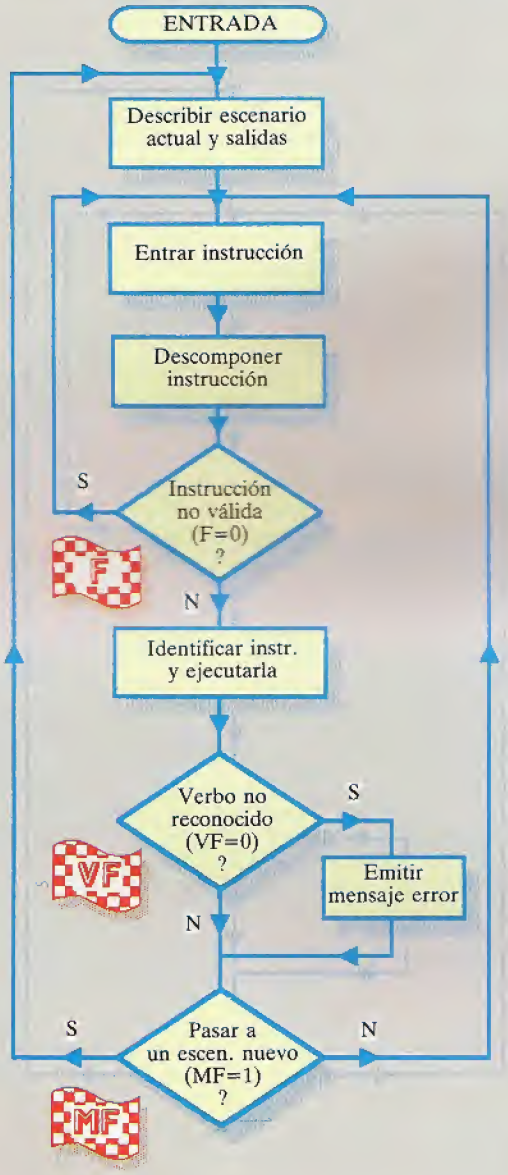
Según el programa, se necesita una velocidad promedio de poco más de 47 millas por horas para completar el viaje con el presupuesto fijado. Cuanto más próxima sea la conjetura al valor verdadero, más pronto hallará *TK!* la solución.

Orden cumplida

En este capítulo veremos cómo nuestro proyecto de programación analiza y obedece las instrucciones que le imparte el jugador

Banderas a cuadros

Las banderas (o *flags* o indicadores) son de uso común en aquellos programas que poseen una construcción modular. Dentro de un módulo se pueden comprobar las condiciones que implican una bifurcación en el control del programa, pero la bifurcación que pueda producirse a raíz del resultado de dicha condición se puede demorar hasta que se realice un retorno a la sección del programa principal. El establecimiento de una variable en un valor preestablecido cuando se efectúa la comprobación, nos permite que el valor de la variable se pueda comprobar más tarde dentro de la sección del programa principal. Las variables que se emplean de esta manera para señalar condiciones se denominan *flags* (o banderas o indicadores). El diagrama de flujo muestra el bucle principal del programa de *El bosque encantado*, tal como lo tenemos construido hasta el momento. La bandera F indica si una instrucción tiene un formato válido o no, y se establece durante la subrutina de "descomposición de instrucciones". La subrutina utilizada para identificar y ejecutar instrucciones normales emplea dos banderas: VF para indicar que la parte del verbo de la instrucción se ha reconocido correctamente. Si al ejecutar una instrucción el jugador se desplaza hasta un escenario nuevo, el hecho de que se ha efectuado un desplazamiento se indica mediante MF. Cuando se comprueba MF dentro del bucle principal del programa, un valor de 1 indica que el bucle ha de bifurcar hacia atrás para describir el nuevo escenario hasta el cual se ha desplazado el jugador



Las aventuras por lo general se construyen de modo tal que el jugador pueda desplazarse de un escenario a otro, recogiendo o abandonando objetos a lo largo del camino. Para llevar a cabo estas tareas hemos empleado estas instrucciones:

AVANZAR	Para desplazarse por los escenarios
RECOGER (objeto)	Para coger un objeto
DEJAR (objeto)	Para deshacerse de un objeto
LISTAR	Para hacer una lista de objetos
MIRAR	Para volver a visualizar la descripción del escenario actual
FIN	Para dar por finalizado el juego

También se podría disponer de variantes de estas instrucciones, tales como IR en vez de AVANZAR, o TOMAR en vez de RECOGER. Parte de la diversión que ofrece un juego de aventuras es determinar cuáles son las palabras que aceptará el juego. Por ejemplo, un jugador podría probar con la instrucción NADAR estando en un escenario seco. Si el programa responde diciéndole al jugador que no puede nadar *aquí*, el jugador podría lógicamente suponer que existen escenarios en los cuales *sí* está permitido nadar. (También existe la posibilidad de que el programador quiera inducir al jugador a pensar que ¡efectivamente es así!)

La cantidad de instrucciones que acepta un juego varía a tenor de la complejidad del juego y de la cantidad de esfuerzo que ha invertido el programador para prever cualquier posible eventualidad. Al crear el programa es esencial que el diseñador se asegure de que el programa no se interrumpa si el jugador trata de entrar una instrucción que no se haya previsto. Todo lo que se requiere es una rutina de autoprotección que imprima "No comprendo", teniendo presente que el programa debe poseer la flexibilidad suficiente para que los jugadores puedan entrar instrucciones de diferentes formas. Por ejemplo, sería muy molesto que un programa aceptara la instrucción TOMAR LAMPARA y respondiera a la instrucción TOMAR LA LAMPARA con un "No comprendo". La incorporación de flexibilidad la analizaremos con mayor profundidad más adelante. Por el momento, hemos de considerar la clase de instrucciones que se podrán impartir durante el juego e idear una rutina que descomponga las mismas de modo tal que puedan ser interpretadas fácilmente.

Descomponer instrucciones

Independientemente de lo que signifique la instrucción, es muy probable que esté construida en modo *imperativo*, como, por ejemplo, AVANZAR SUR HACIA EL RIO o MATAR AL EXTRATERRESTRE. La ventaja de esta estructura de oración es que es fácil de descomponer: el verbo siempre ocupa el primer lugar de la oración, seguido por el objeto del verbo

Ian McKinnell

y, en último lugar, puede haber una cualificación de la acción. La primera etapa del análisis de una instrucción consiste en separar el verbo del resto de la frase. Esta tarea se puede realizar fácilmente explorando la oración carácter a carácter, utilizando MID$, hasta hallar un espacio. La parte de la frase que se halla a la izquierda del espacio es el verbo, y puede ser asignado a la variable VB$. La parte de la frase que hay hacia la derecha puede ser asignada a una segunda variable, NN$. Esta subrutina se utiliza en *El bosque encantado* para descomponer la instrucción contenida en la variable IS$:

```
2500 REM **** S/R DESCOMPONER INSTRUCCION ****
2510 IF IS$="LISTAR" OR IS$="FIN" THEN VB$=IS$:F=1:RETURN
2515 IF IS$="MIRAR" THEN VB$=IS$:F=1:RETURN
2520 F=0
2530 LS=LEN(IS$)
2540 FOR C=1 TO LS
2550 A$=MID$(IS$,C,1)
2560 IF A$<>" " THEN 2590
2570 VB$=LEFT$(IS$,C-1):F=1
2580 NN$=RIGHT$(IS$,LS-C):C=LS
2590 NEXT C
2600
2610 IF F=1 THEN RETURN
2620 PRINT:PRINT"NECESITO AL MENOS DOS PALABRAS"
2630 RETURN
```

La rutina, antes de intentar descomponer la oración, verifica primero que la instrucción no sea ninguna de las tres instrucciones posibles compuestas por una sola palabra, es decir, LISTAR, MIRAR o FIN. Si se trata de una instrucción de una sola palabra, toda ella es asignada a VB$ y se sale de la rutina. Si la instrucción no es ninguna de estas tres, entonces la rutina se introduce en un bucle FOR...NEXT y comienza a explorar en busca del primer espacio. En este bucle se utilizan dos técnicas que merecen una mención especial. Ambas están relacionadas con el hecho de que efectuar un salto condicional para salir fuera de un bucle FOR...NEXT sin pasar por la sentencia NEXT es un estilo de programación sumamente malo. En cambio, para señalar que se ha satisfecho alguna condición (en este caso, que se ha encontrado un espacio) se establece un flag o indicador, F, en uno. En segundo lugar, cuando se ha hallado el primer espacio, es una pérdida de tiempo seguir explorando el resto de la frase.

El bucle se puede terminar limpiamente en este punto estableciendo el contador del bucle, C, en su límite superior, LC. Por consiguiente, cuando el programa vuelva a llegar a NEXT, pasará a la instrucción siguiente, en vez de saltar otra vez hacia atrás hasta la sentencia FOR. Una vez terminado correctamente el bucle, se puede comprobar el estado del indicador F. Un valor de uno en el indicador significa que la frase está compuesta por más de una palabra, y todo cuanto resta por hacer en esta etapa es retornar al bucle principal. Si el indicador no es uno, entonces la instrucción sólo posee una palabra y no es ninguna de las instrucciones de una sola palabra comprobadas previamente. En este caso, antes de retornar en busca de otra instrucción, se imprime un mensaje que señala que se requieren dos palabras.

Instrucciones normales

En la mayor parte del programa el jugador sencillamente se trasladará de un escenario a otro y recogerá o abandonará los objetos que pueda haber encontrado. Por consiguiente, para la mayoría de los escenarios, las instrucciones AVANZAR, RECOGER, DEJAR, LISTAR, MIRAR, FIN (y sus variantes) son suficientes para permitir que el jugador realice esto. Sólo en circunstancias inusuales el jugador deseará utilizar otras instrucciones más especializadas. Por ejemplo, no tiene mayor sentido emplear la instrucción MATAR si no hay nada que poder matar. No obstante, es posible idear una estructura de programa en la cual, en la mayor parte de las ocasiones, sólo se comprueben las seis instrucciones relativas a movimiento y objetos. Cuando el jugador entra en un nuevo escenario, el programa puede verificar si se trata de uno que, por algún motivo, se haya señalado como "especial". Si fuera éste el caso, entonces cualquier nuevo requisito para instrucciones se puede tratar mediante una subrutina de instrucciones específica para ese escenario en particular. Por lo tanto, el bucle principal de llamada de nuestro programa debería hacer lo siguiente:

1) Describir el escenario y listar las salidas.
2) Determinar si éste es "especial".
3) Solicitar una instrucción y, si aquél no es especial, explorar la lista de instrucciones normales.

En el bucle principal ha de haber, asimismo, una facilidad para distinguir entre una instrucción que produce un desplazamiento a un escenario nuevo y otra que no lo produzca. En el primer caso, el bucle necesita volver atrás hasta el comienzo del bucle, para describir el nuevo escenario y decidir si es o no especial. En el segundo caso, sólo es necesario saltar hacia atrás para solicitar una nueva instrucción. La forma más simple de implementar esto consiste en utilizar un "indicador de movimiento", MF, que normalmente está establecido en cero. Si una instrucción implica movimiento, entonces este indicador se establece en uno. El estado de MF se puede comprobar al final del bucle principal, efectuando el salto adecuado. Agréguele a *El bosque encantado* las siguientes líneas:

```
270 GOSUB2500:REM DESCOMPONER INSTRUCCION
275 IF F=0 THEN 260:REM INSTRUCCION NO VALIDA
280 GOSUB3000:REM INSTRUCCIONES NORMALES
290 IF VF=0 THENPRINT:PRINT"NO COMPRENDO"
300 IF MF=1 THEN 240:REM NUEVO ESCENARIO
310 IF MF=0 THEN 260: REM NUEVA INSTRUCCION

3000 REM **** S/R INSTRUCCIONES NORMALES ****
3010 VF=0: REM INDICADOR VERBO
3020 IF VB$="AVANZAR" OR VB$="IR" THEN VF=1:GOSUB3500
3030 IF VB$="RECOGER" OR VB$="TOMAR"THEN VF=1:GOSUB3700
3040 IF VB$="DEJAR" OR VB$="PONER"THEN VF=1:GOSUB3900
3050 IF VB$="LISTAR" OR VB$="INVENTARIO" THEN VF=1:GOSUB4100
3055 IF VB$="MIRAR" THEN VF=1:MF=1:RETURN
3060 IF VB$="FIN" OR VB$="TERMINAR" THEN VF=1:GOSUB4170
3070 RETURN
```

En la primera rutina se utiliza otro indicador, VF, para indicar si el verbo se ha entendido y obedecido o no. VF se establece en uno sólo cuando se ha aislado al verbo. En el bucle principal podemos insertar una sentencia "No comprendo" de autoprotección, comprobando el estado de VF. Si VF permanece en cero, entonces la rutina de análisis no ha reconocido el verbo y se visualiza la sentencia.

En el próximo capítulo del proyecto nos ocuparemos de las subrutinas para recoger, abandonar y listar objetos. De momento, podemos agregarle a nuestro grupo de instrucciones normales una corta subrutina de instrucción FIN:

```
4170 REM **** S/R FIN DEL JUEGO ****
4180 PRINT:PRINT"ESTAS SEGURO (S/N) ?"
4190 GET A$:IF A$<> "S" AND A$<>"N" THEN 4190
4200 IF A$="N" THEN RETURN
4210 END
```

La instrucción MIRAR también es directa. Para volver a describir el escenario actual, simplemente hemos de establecer el "indicador de movimiento", MF, en uno, y retornar al bucle principal del progra-

ma. El establecimiento de MF hará que el programa salte hacia atrás hasta el principio, llamando por tanto a las rutinas que describen un escenario y sus salidas. Dado que el valor de la variable de escenario, P, no sufre ninguna alteración a causa de la instrucción MIRAR, se describirá el mismo escenario. Esta instrucción es útil si, después de que el jugador haya llevado a cabo una serie de acciones, la descripción original del escenario actual se hubiera desplazado fuera de la pantalla.

Añadir flexibilidad

Al impartir instrucciones de movimiento, el jugador puede digitar diferentes formas de la misma instrucción. Por ejemplo, AVANZAR NORTE, IR NORTE y AVANZAR HACIA EL NORTE están solicitando lo mismo. A pesar de que no es de vital importancia que un programa de juego de aventuras reconozca todas estas formas, el hecho de que diversos formatos de instrucción sean legales contribuye a dotar al juego de un mayor interés. Las tres instrucciones de movimiento que acabamos de ofrecer poseen una estructura común: todas empiezan con un verbo de movimiento y la dirección requerida es una palabra discreta. Se puede, por consiguiente, diseñar una rutina que busque la dirección en la parte de la frase que viene después del verbo. La rutina explora esta zona de la oración en busca de espacios, aislando cada palabra y comparándola con las cuatro palabras de dirección buscadas hasta hallar un emparejamiento.

```
3630 REM **** S/R BUSCAR DIRECCION ****
3640 NN$=NN$+" ":LN=LEN(NN$):C=1
3645 FOR I=1 TO LN
3650 IF MID$(NN$,I,1)<>" " THEN NEXT I:RETURN
3655 W$=MID$(NN$,C,I-C):C=I+1
3660 IF W$="NORTE" OR W$="ESTE" THEN NN$=W$:I=LN
3665 IF W$="SUR" OR W$="OESTE" THEN NN$=W$:I=LN
3670 NEXT I
3675 RETURN
```

En el capítulo anterior del proyecto desarrollamos una rutina de movimiento. Para añadir esta nueva rutina a la de movimiento sólo hemos de agregar la siguiente línea:

```
3505 GOSUB 3630:REM BUSCAR DIRECCION
```

Vale la pena destacar que esta rutina no obedecerá instrucciones tales como AVANZAR EN UNA DIRECCION NORTEÑA, puesto que la rutina no puede aislar la palabra de dirección. Sería posible diseñar una rutina que trabajara en función del principio de explorar grupos de cuatro y cinco letras, comparando cada grupo con las cuatro palabras de dirección posibles. Sin embargo, tal rutina tomaría mucho tiempo de ejecución. Por otra parte, nuestro programa sí aceptará AVANZAR NORTEÑA, dado que la rutina de movimiento finalmente utiliza la primera letra de la segunda parte de la oración, NN$. En este caso, la N de NORTEÑA se aceptaría como si se tratara de la N de NORTE.

Listados Digitaya

```
1220 GOSUB1700:REM ANALIZAR INSTRUCCIONES
1225 IF F=0 THEN 1210:REM INSTRUCCION NO VALIDA
1230 GOSUB 1900:REM INSTRUCCIONES NORMALES
1240 IF VF=0 THEN PRINT"NO COMPRENDO"
1250 IF MF=1 THEN 1160:REM NUEVA POSICION
1260 IF MF=0 THEN 1210:REM NUEVA INSTRUCCION
1700 REM **** S/R ANALIZAR INSTRUCCION ****
1705 F=0:REM BANDERA CERO
1710 IF IS$="FIN" OR IS$="LISTAR" THEN VB$=IS$:F=
     1:RETURN
1720 IF IS$="MIRAR" THEN VB$=IS$:F=1:RETURN
1730 :
1740 REM ** DESCOMPONER INSTRUCCION **
1750 VB$="":NN$="":REM VERBO Y SUSTANTIVO CERO
1770 LS=LEN(IS$)
1780 FOR C=1 TO LS
1790 A$=MID$(IS$,C,1)
1800 IF A$=" " THEN VB$=LEFT$(IS$,C-1):NN$=RIGHT$
     (IS$,LS-C):F=1:C=LS
1810 NEXT
1830 IF F=0 THEN PRINT:PRINT"NECESITO AL MENOS DOS
     PALABRAS"
1840 RETURN
1850 :
1900 REM **** S/R ACCIONES NORMALES ****
1910 VF=0
1920 PRINT
1930 IF VB$="AVANZAR" OR VB$="IR" THEN VF=1:GOSUB2000
1940 IF VB$="RECOGER" OR VB$="TOMAR" THEN VF=
     1:GOSUB2140
1950 IF VB$="DEJAR" OR VB$="PONER" THEN VF=1:GOSUB2360
1960 IF VB$="LISTAR" OR VB$="INVENTARIO" THEN
     VF=1:GOSUB2540
1965 IF VB$="MIRAR" THEN VF=1:MF=1:RETURN
1970 IF VB$="FIN" OR VB$="TERMINAR" THEN VF=1:GOSUB2610
1980 RETURN

2015 GOSUB8600:REM BUSCAR DIRECCION
2610 REM **** S/R FIN DEL JUEGO ****
2620 PRINT:PRINT"ESTAS SEGURO (S/N) ?"
2630 GETA$:IFA$<>"S" AND A$<>"N"THEN2630
2640 IFA$="N"THEN RETURN
2650 END

8600 REM **** S/R BUSCAR DIRECCION ****
8610 NN$=NN$+" ":LN=LEN(NN$):C=1
8620 FOR I=1 TO LN
8630 IF MID$(NN$,I,1)<>" " THEN NEXT I:RETURN
8640 W$=MID$(NN$,C,I-C):C=I+1
8650 IF W$="NORTE" OR W$="ESTE" THEN NN$=W$:I=LN
8660 IF W$="SUR" OR W$="OESTE" THEN NN$=W$:I=LN
8670 NEXT I
8680 RETURN
```

Complementos al BASIC

Spectrum:
En ambos programas, utilice I$ por IS$, B$ por VB$ y R$ por NN$.
En *Digitaya*, reemplace las siguientes líneas:

```
1790 LET A$=I$(C TO C)
1800 IF A$="" THEN LET B$=I$(TO C-1):LET
     R$=I$ (LEN(I$)-LS+C+1 TO): LET F=
     1:LET C=LS
2630 LET A$=INKEY$:IF A$<>"S"
     AND A$<>"N" THEN 2630
8630 IF R$(I TO I)<>"" THEN NEXT I:RETURN
8640 LET W$=R$(C TO I-1):LET C=I+1
```

En *El bosque encantado* sustituya estas líneas:

```
2550 LET A$=I$(C TO C)
2570 LET B$=I$(TO C-1):LET F=1
2580 LET R$=I$(LEN(I$)-LS+C+1 TO):LET C=LS
3650 IF R$(I TO I)<>"" THEN NEXT I:RETURN
3655 LET W$=R$(C TO I-1):LET C=I+1
4190 LET A$=INKEY$:IF A$<>"S" AND
     A$<>"N" THEN GOTO 4190
```

BBC Micro:
En *Digitaya* reemplace esta línea:

```
2630 REPEAT:A$=GET$:UNTIL A$="S"
     OR A$="N"
```

y ésta en *El bosque encantado*:

```
4190 REPEAT:A$=GET$:UNTIL A$="S"
     OR A$="N"
```

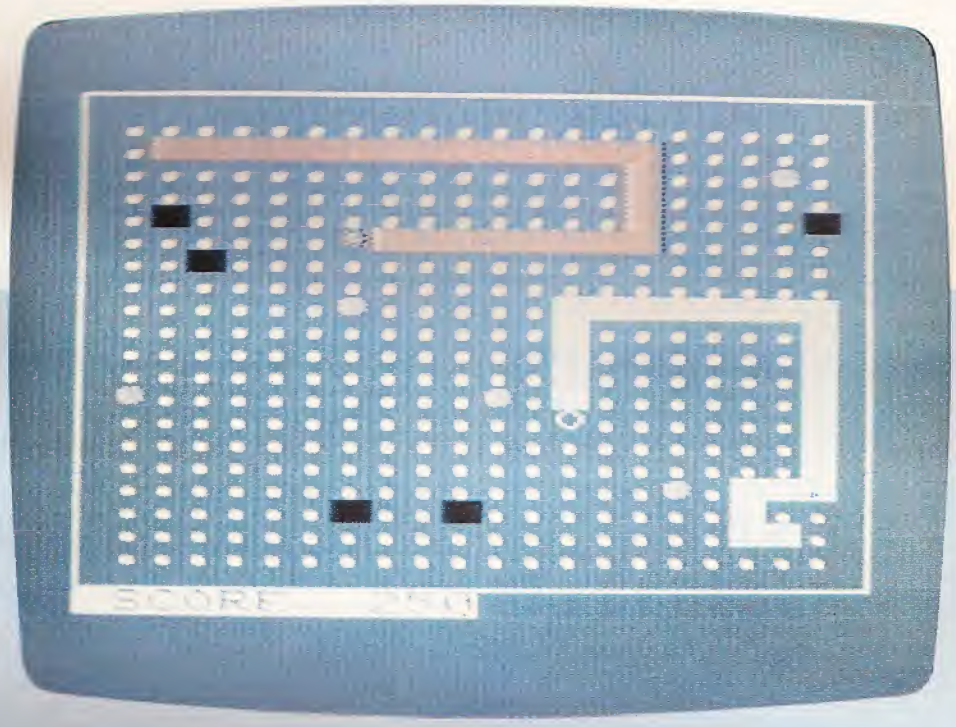
La trampa

¡Atención a la corriente eléctrica!: ella constituye "la trampa". El listado de este interesante juego ha sido escrito para el Vic-20

En este juego, usted se desplaza dentro de un rectángulo dejando una estela tras de sí. El ordenador controla un robot que, al trasladarse de lugar, deja asimismo un trazo a sus espaldas. Si usted toca los límites del rectángulo o si corta un trazo, se electrocutará.

Los puntos se cuentan de la manera siguiente: Si el robot se detiene (el círculo verde): 200 puntos; si usted pasa sobre un punto malva: 50 puntos; si usted pasa sobre un punto blanco: 10 puntos.

La velocidad en la ejecución del juego aumenta en cada partida.



```
10 RESTORE
13 GOSUB3000:GOSUB2000
15 X=RND(-TI):DEFFNR(X)=(PEEK(7663+X)=3)
20 D(0)=1:D(1)=22:D(2)=-1:D(3)=-22
30 L=150+(KJ$="J")*50
40 PP=46:PC=437:DP=1:DC=2:SC=7680:CL=38400:POKE36878,15
45 GOSUB1000
50 IFKJ$="J"THEN62
55 GETA$:POKE198,0
60 ND=22*((A$="P")-(A$="."))+(A$="L")-(A$=":"):IFND<>0THENDP=ND
61 GOTO70
62 SYS7600:X1=0:FORX=0TO3:X1=X1-FNR(X):NEXT:IFX1>1THEN70
65 ND=22*(FNR(0)-FNR(1)-FNR(2)-FNR(3):IFND<>0THENDP=ND
70 NP=PP+DP:X1=PEEK(SC+NP):IFX1<>46THEN700
75 POKE36874,128:S=S+10
80 POKESC+PP,160:POKECL+PP,2:POKESC+NP,42:POKECL+NP,2:PP=NP
85 PRINT"■■■■■■■■■■■■■■■■■■■■■■■■PTS "S" ■ ";:POKE36874,0:POKE36876,0
90 NC=PC+D(DC):IFPEEK(SC+NC)<>460RRND(1)<.15THEN200
100 POKESC+PC,160:POKECL+PC,5:POKESC+NC,15:POKECL+NC,5:PC=NC
110 FORX=0TOL:NEXT:GOTO50
200 FORX=1TO19:DC=INT(RND(1)*4):IFPEEK(SC+PC+D(DC))=46THENNC=PC+D(DC):GOTO100
210 NEXT
220 S=S+200:L=L/2
230 FORX=1TO4:FORY=128TO160:POKE36875,Y:NEXT:NEXT:POKE36875,0:POKE198,0:GOTO40
700 IFX1=81THENPOKE36876,200:S=S+50A:GOTO80
800 POKE36877,250:FORX=1TO2000:NEXT:POKE36877,0
810 PRINT"■■■■■■■■■■■■■■■■■■■
```

Toque de distinción

He aquí un dispositivo para diseñar gráficos que se destaca especialmente porque puede ser utilizado por la mayoría de ordenadores personales

Todos los ordenadores de mayor venta en la actualidad ofrecen visualizaciones de gráficos en alta resolución. Sin embargo, a menos que se disponga de software para gráficos ya escrito, la creación de tales visualizaciones exige mucho tiempo y esfuerzo y muchas de las características no se utilizan al completo. Un programa para dibujar "bocetos" resulta insuficiente, porque a menudo el usuario desea copiar en el ordenador una imagen ya existente en lugar de simplemente dibujar a mano alzada.

Con esta finalidad se han comercializado varios digitalizadores, pero casi todos se han diseñado básicamente para ser utilizados con máquinas específicas, como el BBC Micro o el ZX Spectrum. La tablilla para gráficos Touchmaster está diseñada para trabajar con una amplia gama de máquinas personales (algunas de las cuales requerirán un cable o una interface apropiada). Este dispositivo se está promocionando, asimismo, como teclado de recambio, pero la simplicidad de su diseño implica que dicha utilización se ve limitada a la selección de opciones de menú o al control de juegos. Para la entrada de datos, así como para la carga del propio software del Touchmaster, sigue siendo necesario un teclado de ordenador.

El Touchmaster viene instalado en una carcasa plástica de color gris que mide 350×330×35 mm. La parte posterior de la misma está ligeramente elevada, formando un ángulo adecuado para dibujar. Se suministra con un transformador y un LED rojo que indica si el dispositivo está conectado o no. Para que la tablilla se pueda utilizar con una amplia gama de máquinas personales, en el panel posterior hay instalados conectores para interfaces tanto en serie como en paralelo, junto con un conector (que no se menciona en los manuales) para un interruptor de pie. En realidad, los manuales no son muy adecuados: el de hardware ofrece instrucciones para la conexión de la tablilla y proporciona algunos programas sencillos en BASIC para leer coordenadas, pero carece de los suficientes detalles.

La tablilla se basa en la tecnología de membrana desarrollada en los teclados del ZX81 y del Spectrum, y ofrece una resolución de 256×256 pixels. La capa superior está separada de la película sensitiva inferior por un tejido aislante, y la presión efectuada sobre la capa superior la obliga a hacer contacto con la película. La tablilla contiene un microprocesador que explora la película sensitiva en una dirección, mientras explora al mismo tiempo la capa inferior en otra dirección, y la coordenada del "punto de contacto" se envía entonces a través de interfaces tanto en serie como en paralelo. La interface en serie se utiliza para conectar la tablilla al BBC Micro, mientras que la interface en paralelo es necesaria para emplearla con el Commodore 64, el Vic-20, el Spectrum y el Dragon. La resolución del Touchmaster es inferior a la que proporcionan muchas visualizaciones en pantalla de alta resolución, de modo que los usuarios del BBC Micro, por ejemplo, no podrán resolver un pixel individual en Mode 0.

Ian McKinnell

El teclado del Touchmaster
La hoja que se proporciona con el software se encaja simplemente en la superficie de dibujo. La instrucción se ejecuta en la pantalla al hacer presión sobre la instrucción apropiada en el lado derecho de la cubierta y mover el lápiz, el punzón o el dedo hasta la superficie de dibujo

Programa Multipaint

Con el Touchmaster se proporciona un programa de dibujo llamado *Multipaint*. Éste ofrece una demostración de las facilidades disponibles, pero no representa una gran ayuda para los gráficos. Una plantilla plástica proporciona un menú de las facilidades, con la opción elegida visualizada en una ventana de "estado" en la parte inferior de la pantalla. Se pueden utilizar cinco tipos distintos de pincel; el ancho de cada uno puede oscilar entre 2 y 32 pixels, en pasos de dos pixels. La ventana muestra,

Primer paso: esbozo

Segundo paso: sombreado

Tercer paso: detalles

Cuarto paso: color

Las etapas de una escena

El Touchmaster se puede utilizar como tablilla para gráficos con el software Multipaint y la cubierta que se proporciona con el producto. En las fotografías vemos las etapas de una escena, desde el bosquejo hasta la imagen coloreada y sombreada

asimismo, la modalidad en curso de dibujo (Dots, Points o Freehand: pixels, puntos o mano alzada) y los colores seleccionados para primer plano y fondo. Éstos se pueden cambiar pulsando la opción requerida del menú hasta que aparezca en la ventana de estado.

Una vez se han seleccionado los colores y los tipos de pincel, hay otras opciones disponibles para la creación de cajas, círculos, polígonos y líneas "elásticas". Con el paquete se proporciona un punzón, pero también se puede utilizar la presión de los dedos. El Touchmaster, con la gran superficie de su tablilla, no se ve afectado por las mismas restricciones del Koala-pad; la presión de un dedo se puede traducir a una coordenada exacta y no a una mera aproximación, ¡de modo que el dibujo electrónico a dedo es realmente posible!

Lamentablemente, el Touchmaster no ofrece más que facilidades rudimentarias. Hay una opción FILL (rellenar) marcada en la plantilla y documentada en los manuales, pero (al menos en el Spectrum) la misma no parece funcionar de la forma esperada. Tampoco hay una facilidad para ampliación ni para edición, lo que significa que los colores no se pueden cambiar. En el Spectrum, con el cual suele ser más fácil dibujar en blanco y negro antes de añadir color, esto constituye una evidente desventaja.

Como elemento de hardware, la tablilla Touchmaster está en una posición mucho mejor frente a rivales como el Grafpad y el Koala-pad. Es de construcción sólida y ofrece una superficie de dibujo de tamaño A4 que se puede conectar a la mayoría de los ordenadores personales más populares. Una ventaja significativa es que si en el futuro usted decide ampliar su máquina o directamente cambiarla por otra, todo lo que se requiere es una nueva interface; además, por supuesto, del software adecuado.

Resulta decepcionante que la documentación y el software suministrados sean tan pobres en comparación con el estándar de la tablilla. El Touchmaster está dando origen a una gama de software diseñado específicamente para usar con esta tablilla, si bien la verdadera prueba de su éxito se producirá cuando las casas de software independientes decidan apoyarla.

TOUCHMASTER MSX

DIMENSIONES

350×330×35 mm

INTERFACES

Se proporcionan puertas en serie y en paralelo para utilizar con una vasta gama de ordenadores personales o con el propio teclado del Touchmaster

DOCUMENTACION

Si bien los manuales contienen la información necesaria para utilizar el Touchmaster, hubiera sido de agradecer una información más detallada

VENTAJAS

La amplia gama de interfaces, junto con los diferentes paquetes de software, hacen del Touchmaster un periférico sumamente adaptable

DESVENTAJAS

La calidad del software queda en una posición muy precaria si se la compara con la de dispositivos similares; los manuales son limitados

De colores

Existen diversas utilidades y programas para juegos, de entretenimiento y educativos a la venta para el Touchmaster en las principales tiendas especializadas. Cada paquete incluye el software apropiado (para una gama de micros), instrucciones e imaginativas cubiertas, algunas de las cuales se pueden apreciar aquí

¿Quién lo hizo?

Mostramos la preparación de una base de datos sencilla en un ejemplo que propone la investigación de un crimen

En una pequeña localidad de los Andes se ha perpetrado un asesinato. Zacarías ha sido atacado con un hacha y muerto. Sabemos que tanto Mateo como José poseen hachas, que Jaime y Esteban poseen escopetas y que la prima Juana tiene un cuchillo. Mateo y Jaime tenían sangre en las manos cuando el comisario local los interrogó.

Nuestra base de datos en LOGO sobre este crimen consistirá en una lista de *hechos*, cada uno de los cuales consistirá en una *relación*, junto con uno o más sustantivos. Al representarlo en LOGO, un hecho es [POSEE MATEO HACHA] o, en correcto castellano, "Mateo posee un hacha". Para representar el hecho de que Jaime tenía sangre en las manos utilizamos [ENSANGRENTADO JAIME].

Comenzamos nuestra investigación con una base de datos vacía:

```
TO PREPARACION
  MAKE "BASEDATOS []
END
```

Luego vamos añadiendo hechos a nuestra base de datos a medida que los vamos descubriendo (siempre y cuando no estuvieran ya en ella). Por ejemplo, entraríamos AGREGAR [POSEE JUANA CUCHILLO] empleando este procedimiento:

```
TO AGREGAR :HECHO
  IF NOT MEMBER? :HECHO :BASEDATOS THEN
  MAKE "BASEDATOS FPUT :HECHO
  :BASEDATOS
END
```

La base de datos finalmente estará llena con:

```
[[ENSANGRENTADO MATEO] [ENSANGRENTADO
JAIME] [ASESINADO
  ZACARIAS HACHA] [POSEE MATEO HACHA]
[POSEE JOSE HACHA] [POSEE JAIME ESCOPETA]
[POSEE ESTEBAN ESCOPETA] [POSEE JUANA
  CUCHILLO]]
```

Para imprimir la base de datos utilice el procedimiento MOSTRAR. El mismo puede ir seguido ya sea de "TODO, en cuyo caso se imprimirá toda la base de datos, o bien del nombre de una relación, en cuyo caso sólo se imprimirán los hechos de esa relación. Por consiguiente, MOSTRAR "POSEE nos mostrará quién posee qué.

```
TO MOSTRAR :S
  IF :S="TODO THEN LISTAR.TODO :BASEDATOS
  LISTAR.REL :S :BASEDATOS
END

TO LISTAR.TODO :LISTA
  IF EMPTY? :LISTA THEN STOP
  PRINT FIRST :LISTA
  LISTAR.TODO BUTFIRST :LISTA

END
TO LISTAR.REL :S :LISTA
  IF EMPTY? :LISTA THEN STOP
  IF :S=FIRST FIRST :LISTA THEN PRINT FIRST
    :LISTA
  LISTAR.REL :S BUTFIRST :LISTA
END
```

Ahora debemos pensar en formas de interrogar a la base de datos. La forma de interrogación más sencilla consiste en comprobar si se sabe que un hecho es verdadero. Esto lo hacemos mediante un procedimiento llamado PREG, que comprueba si un hecho está incluido en la base de datos. Por ejemplo, PREG [POSEE JUANA CUCHILLO] daría como respuesta SI.

```
TO PREG :HECHO
  IF MEMBER? :HECHO :BASEDATOS PRINT "SI
    ELSE PRINT "NO
END
```

Para nuestra investigación sería mucho más útil que pudiéramos formular preguntas como "¿Quién posee un hacha?". Esto lo manejaremos empleando "variables". Se dará por sentado que toda palabra cuyo primer carácter sea un ? es una variable. Entonces podemos parafrasear la pregunta así:

```
CUAL [POSEE ?ALGUIEN HACHA]
```

La respuesta a esto sería una lista de todos los valores posibles de la variable ?ALGUIEN que sean coherentes con la información de la base de datos.

```
[?ALGUIEN MATEO]
[?ALGUIEN JOSE]
NO (MAS) RESPUESTAS
```

Podemos tener múltiples variables. Por ejemplo:

```
CUAL [MATO ?HOMBRE ?UTENSILIO]
```

dará la respuesta:

```
[?HOMBRE ZACARIAS] [?UTENSILIO HACHA]
NO (MAS) RESPUESTAS
```

Consideremos individualmente los procedimientos que hacen viable este análisis de la base de datos. CUAL le pasa la tarea a HALLAR, indicando BASEDATOS como la fuente de hechos.

```
TO CUAL :INTERROGACION
  HALLAR :INTERROGACION :BASEDATOS
  PRINT [NO (MAS) RESPUESTAS]
END
```

HALLAR crea dos variables globales, VARS y RESP: VARS se utiliza para retener cada posible conjunto de valores de las variables de la pregunta, y éstos se reúnen en la lista RESP.

```
TO HALLAR :INTERROGANTES :DATOS
  MAKE "VARS []
  MAKE "RESP []
  COMPARAR :INTERROGACION :DATOS
  PRINTL :RESP
END
```

COMPARAR examina cada uno de los hechos de la base de datos de uno en uno. De haber un emparejamiento, el nuevo conjunto de valores de VARS se agrega a RESP antes de volver a colocar a VARS en la lista vacía. COMPARAR sigue entonces trabajando en la base de datos para ver si existe alguna otra posible pareja.

```
TO COMPARAR :INTERROGACION :DATOS
  IF EMPTY? :DATOS THEN STOP
  IF PAREJA? :INTERROGACION FIRST :DATOS
  THEN MAKE "RESP FPUT :VARS :RESP
  MAKE "VARS []
  COMPARAR :INTERROGACION BUTFIRST :DATOS
END
```

Para apreciar lo que hace PAREJA? considere el caso en el cual las entradas sean [POSEE ?ALGUIEN HACHA] y [POSEE JOSE HACHA], en respuesta a las cuales PAREJA? producirá TRUE y establecerá VARS en [?ALGUIEN JOSE]. Si las entradas fueran [POSEE ?ALGUIEN HACHA] y [MATO ZACARIAS HACHA], entonces PAREJA? produciría FALSE.

Las verdaderas dificultades surgen, sin embargo, cuando hay implicada más de una variable. Se emplea VALOR? para verificar si la variable ya se le ha asignado un valor para ese hecho en la base de datos.

Aquí hemos utilizado una notación alternativa para las condiciones del LOGO. TEST evalúa una condición. Si el resultado es verdadero, se llevan a cabo las acciones que van después de IFTRUE; de lo contrario, se efectúan las acciones que siguen a IFFALSE.

```
TO PAREJA? :INTERROGACION :HECHO
  IF ALLOF EMPTY? :INTERROGACION EMPTY?
    :HECHO THEN OUTPUT "TRUE
  TEST FIRST FIRST :INTERROGACION="?
  IFTRUE IF NOT VALOR? FIRST :INTERROGACION
    FIRST :HECHO :VARS THEN OUTPUT "FALSE
  IFFALSE IF NOT (FIRST  :INTERROGACION=
    FIRST :HECHO) THEN OUTPUT "FALSE
    OUTPUT PAREJA? BUTFIRST
    :INTERROGACION
  BUTFIRST :HECHO
END
```

Para ver cómo funciona VALOR?, consideremos en primer lugar el caso en el que las entradas sean ?UTENSILIO, HACHA y [?HOMBRE ZACARIAS]. VALOR? intenta determinar si la variable ?UTENSILIO podría tener el valor HACHA. Existen tres posibilidades: ?UTENSILIO ya posee un valor, que no es HACHA, y VALOR? produce FALSO; ?UTENSILIO ya posee el valor HACHA, y VALOR? produce VERDADERO; o ?UTENSILIO no posee ningún valor, de modo que se le da el valor HACHA, y se agrega esta información a VARS y se produce VERDADERO.

```
TO VALOR? :NOMBRE :VALOR :VLISTA
  IF EMPTY? :VLISTA THEN MAKE "VARS LPUT
    LIST :NOMBRE :VALOR :VARS OUTPUT
    "TRUE
  TEST :NOMBRE=FIRST FIRST :VLISTA
  IFTRUE IF :VALOR=LAST FIRST :VLISTA THEN
    OUTPUT "TRUE ELSE OUTPUT "FALSE
  OUTPUT VALOR? :NOMBRE :VALOR BUTFIRST
    :VLISTA
END
```

PRINTL hace que los componentes de RESP se impriman uno debajo del otro.

```
TO PRINTL :LISTA
  IF EMPTY? :LISTA STOP
  PRINT FIRST :LISTA
  PRINTL BUTFIRST :LISTA
END
```

Interrogatorios más complejos

Nuestra investigación no llegará muy lejos, sin embargo, a menos que podamos formular preguntas más complejas, tales como "¿Con qué utensilio fue asesinado Zacarías y quién posee un utensilio como ése?". En LOGO, esto se lee:

```
CUAL [[MATO ZACARIAS ?UTENSILIO]
  [POSEE ?SOSPECHOSO ?UTENSILIO]]
```

Ahora CUAL toma como entrada una lista de preguntas, y los valores hallados serán aquellos que hagan que todas ellas resulten verdaderas. Entonces, si se desea formular una única pregunta con esta nueva forma de CUAL, la sintaxis que utilizamos es:

```
CUAL [[POSEE ?ALGUN CUCHILLO]]
```

En estos procedimientos sólo es necesario introducir unas ligeras modificaciones:

```
TO CUAL :INTERROGANTES
  HALLAR :INTERROGANTES :BASEDATOS
  PRINT [NO (MAS) RESPUESTAS]
END

TO HALLAR :INTERROGANTES :DATOS
  MAKE "VARS []
  MAKE "RESP []
  COMPARAR :INTERROGANTES :DATOS
  PRINTL :RESP
END
```

Ahora COMPARAR tiene que realizar una tarea bastante difícil. Tomemos como ejemplo la entrada [[MATO ZACARIAS ?UTENSILIO] [POSEE ?SOSPECHOSO ?UTENSILIO]]. COMPARAR revisa la base de datos, de hecho en hecho, en busca de una pareja para el primer interrogante, y acaba emparejando ?UTENSILIO con HACHA. La rutina considera entonces el segundo interrogante ([POSEE ?SOSPECHOSO ?UTENSILIO]), empezando otra vez por el comienzo de la base de datos. Se encuentra una pareja para la segunda condición, con el valor para ?UTENSILIO de HACHA y el de MATEO para ?SOSPECHOSO. No hay ningún otro interrogante, de modo que ésta es una posible solución.

Pero todavía no hemos terminado; puede haber otros valores que satisfagan el segundo interrogante, manteniendo a ?UTENSILIO como HACHA. De modo que comparar prosigue revisando la base de datos a partir del lugar en que la dejó, y realmente encuentra una segunda solución para ?SOSPECHOSO con JOSE. Por supuesto, el procedimiento no se detiene allí, sino que continúa buscando en la BASEDATOS. En esta ocasión llega hasta el final sin haber encontrado ningún otro valor emparejado.

Es posible, no obstante, que exista otra solución para el primer interrogante distinta de HACHA para ?UTENSILIO, de manera que hemos de volver atrás hasta el punto de la base de datos en el que encontramos la pareja y continuar desde allí. Este proceso se denomina *vuelta atrás (backtracking)*. En este

David Higham

Enigma de un crimen

El señor Harcourt fue hallado en la parte trasera de una furgoneta con 30 puñaladas hechas con un cincel. La policía identificó a cuatro sospechosos: un constructor, un carnicero, una jardinera y un diseñador. Cada uno de ellos tuvo acceso a instrumentos punzantes: un cuchillo (el carnicero), tijeras (la jardinera), un cortador (el diseñador) y un cincel (el constructor). Uno de ellos fue visto en una esquina, otro en el cobertizo de un jardín, y un tercero aseguró haber estado acostado, durmiendo. El que pueda ser relacionado con la furgoneta es el asesino. La investigación condujo a lo siguiente: **1)** La noche del crimen, ningún sospechoso estaba en posesión de su herramienta; **2)** Al carnicero se le vio en el cobertizo abriendo cartas con el cortador; **3)** Un testigo presencial confirmó que el constructor se hallaba parado en una esquina, donde la jardinera encontraría luego las tijeras que le faltaban; **4)** La jardinera estaba acostada, y utilizó un cuchillo de carnicero para prepararse un bocadillo

caso, en realidad no existe ninguna otra solución.

Para no perder la pista del punto en que se encuentra en su asignación de variables, COMPARAR coloca los valores actuales en una *pila* antes de utilizar PAREJA? (puesto que PAREJA? podría alterar estos valores), y posteriormente los recupera. Éste es el procedimiento completo:

```
TO COMPARAR :INTERROGANTES :DATOS
  IF EMPTY? :INTERROGANTES THEN MAKE
    "RESP FPUT :VARS :RESP STOP
  IF EMPTY? :DATOS THEN STOP
  EMPUJAR :VARS
  TEST PAREJA? FIRST :INTERROGANTES FIRST
    :DATOS
  IFTRUE COMPARAR BUTFIRST :INTERROGANTES
    :BASEDATOS
  TIRAR "VARS
  COMPARAR :INTERROGANTES BUTFIRST :DATOS
END
```

En COMPARAR utilizamos una pila para seguir la pista del valor de VARS, en lugar de una variable temporal, debido a que COMPARAR se llamaría a sí misma entre el momento en que deseáramos guardar los valores y el momento en que deseáramos recuperarlos. Por consiguiente, cualquier variable temporal de este tipo sería destruida en la siguiente llamada y se perderían los valores originales. La pila impide que suceda esto.

EMPUJAR coloca un valor "encima" de la pila, creando primero la variable PILA si la misma no existiera aún.

```
TO EMPUJAR :DATOS
  IF NOT THING? :PILA THEN MAKE "PILA []
  MAKE "PILA FPUT :DATOS :PILA
END
```

TIRAR toma un elemento de la pila y lo asigna a una variable.

```
TO TIRAR :NOMBRE
  MAKE :NOMBRE FIRST :PILA
  MAKE "PILA BUTFIRST :PILA
END
```

Lo que tenemos ahora son los rudimentos de un lenguaje de "programación lógica". Éste es un lenguaje en el que simplemente se añaden hechos y reglas a una base de datos y luego se interroga a esa base de datos mediante descripciones lógicas de los datos que se requieren. Hasta la fecha, el mejor ejemplo de lenguaje de programación lógico es el PROLOG; ¡pero ésa es otra historia!

Complem. al LOGO

Algunas versiones de LOGO MIT no poseen EMPTY? ni MEMBER? Ya hemos ofrecido definiciones para ellas en anteriores *Complementos al* LOGO.

En todas las versiones LCSI utilice EMPTYP por EMPTY? y MEMBERP por MEMBER? Existe una primitiva, EQUALP, que comprueba si sus dos entradas son iguales. Empléela para comparar listas y palabras en lugar del signo de igualdad (que funciona para listas sólo en algunas versiones LCSI).

La sintaxis IF del LOGO LCSI queda demostrada de la siguiente manera:

```
IF EMPTYP :CONTENIDO
[PRINT[NADA ESPECIAL]]
[PRINT :CONTENIDO]
```

Si la condición es verdadera se lleva a cabo la primera lista después de la condición, y la segunda si es falsa.

El LOGO LCSI también admite la sintaxis TEST, IFTRUE, IFFALSE para las condiciones

Paso a paso

Estudiaremos en este capítulo los motores paso a paso, usados de manera preferencial en el control de los movimientos de todo tipo de robots

La construcción de un motor paso a paso es muy diferente de la de un motor normal. Para comprender sus principios de funcionamiento, veremos en primer lugar cómo trabaja un motor paso a paso simplificado.

En nuestro ejemplo (véase el diagrama titulado "Un paso cada vez" en la página siguiente) hay dos juegos de bobinas ("a" y "b") en el estator y dos pares de polos electromagnéticos en el rotor. En los motores que utilizamos en la construcción de nuestro robot se incluye un mayor número de bobinas de estator y de polos de rotor que los que aparecen en el ejemplo.

El único problema de esta conveniente forma de control de motor es que el motor consume tanta corriente cuando está parado como cuando está en movimiento.

Por otra parte, no es posible hacer girar el motor a gran velocidad: las distintas bobinas no se pueden activar y desactivar con la rapidez suficiente. No obstante, ninguno de estos problemas tiene importancia en nuestra aplicación.

Nuestro motor simplificado sólo puede girar en pasos de 45°. Además, no se puede controlar la dirección de la rotación. Los motores utilizados en el robot, sin embargo, poseen cuatro juegos de bobinas que se activan por pares, y el rotor, asimismo, posee muchas más bobinas de las que muestra nuestro ejemplo. Ello significa que sí se puede controlar la dirección de rotación y que el ángulo de paso se reduce a 7,5°. Para conseguir este giro por pasos tan preciso, se deben activar las cuatro bobinas en una secuencia dada y compleja, del siguiente modo:

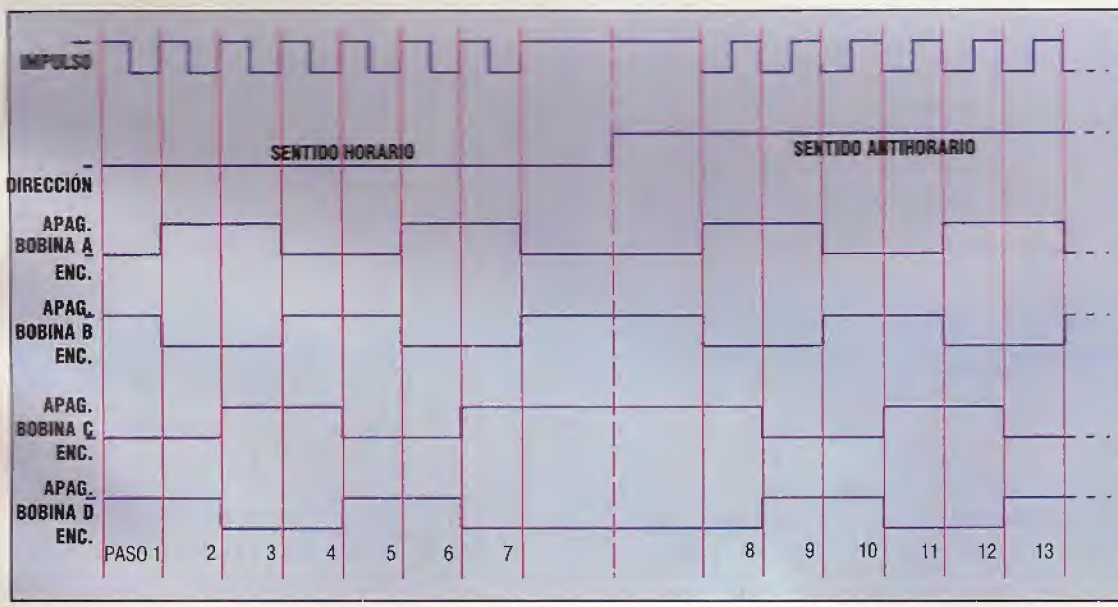
Tabla de la secuencia de activación

Paso	Bobina A	Bobina B	Bobina C	Bobina D
1	encendida	apagada	encendida	apagada
2	apagada	encendida	encendida	apagada
3	apagada	encendida	apagada	encendida
4	encendida	apagada	apagada	encendida
5	encendida	apagada	encendida	apagada

Esta secuencia de activación se podría gestionar mediante software, utilizando cuatro bits de la puerta para el usuario para controlar las cuatro bobinas. Sin embargo, ello requiere una programación complicada y, ciertamente, el BASIC no produciría estas secuencias de control con la rapidez suficiente. Un método más sencillo consiste en emplear un chip que ha sido diseñado especialmente para el control de motores paso a paso: el SAA 1027. Éste contiene los activadores de salida y todos los circuitos lógicos para activar las bobinas en el orden correcto necesario para activar un motor paso a paso.

Para hacer que el motor efectúe un giro de un paso, se requiere un único impulso proveniente de la puerta para el usuario, siendo necesaria aún otra línea de señal para determinar la dirección de la rotación. El chip contiene etapas de entrada para detectar los cambios en las tres entradas: un impulso para hacer que el motor gire un paso, una entrada de inicialización y una entrada de dirección que invierte la secuencia activadora de las bobinas del estator. Las entradas van a parar a un circuito contador bidireccional para producir la correcta secuencia de salidas hacia las bobinas del estator.

Por último, el chip también contiene una etapa activadora de salida de potencia que puede manipular hasta 550 mW. La inclusión de esta etapa sig-



Activando las bobinas

El diagrama muestra cómo las cuatro bobinas se energizan en secuencia, produciendo en el rotor un movimiento en sentido horario. Después de haberse completado los pasos 1, 2, 3 y 4 la secuencia se reinicia, activando el paso 5 la misma combinación de bobinas que el paso 1. Cuando cambia la señal de dirección, la secuencia se invierte. El paso 9 activa las bobinas en la misma combinación que el paso 7; el paso 10 energiza las bobinas de la misma forma que el paso 6, y así sucesivamente, haciendo que el rotor gire en sentido antihorario, siendo la posición del rotor después del paso 8 la dirección más extrema en el sentido horario

nifica que el motor se puede conectar directamente al chip sin que exista necesidad de transistores de potencia externos.

La complejidad del chip activador de motores paso a paso permite que el resto del circuito necesario para el control del robot sea verdaderamente muy simple. Cada motor requiere uno de estos chips, al cual se conecta el motor. Lamentablemente, los chips activadores operan a un voltaje de alrededor de 12 V, mientras que la puerta para el usuario de su ordenador opera a 5 V. Es decir, un 0 lógico es 0 V (más o menos) y un 1 lógico es 5 V. Las entradas del chip activador exigen 0 V para una entrada 0 y entre 7,5 y 12 V para un 1. Para la conexión en interface de la puerta para el usuario con los chips activadores también precisaremos un chip buffer especial de dos voltajes con las entradas operando en un voltaje y las salidas en otro. Éste es el chip 40109, también necesario para el circuito.

Lista de componentes

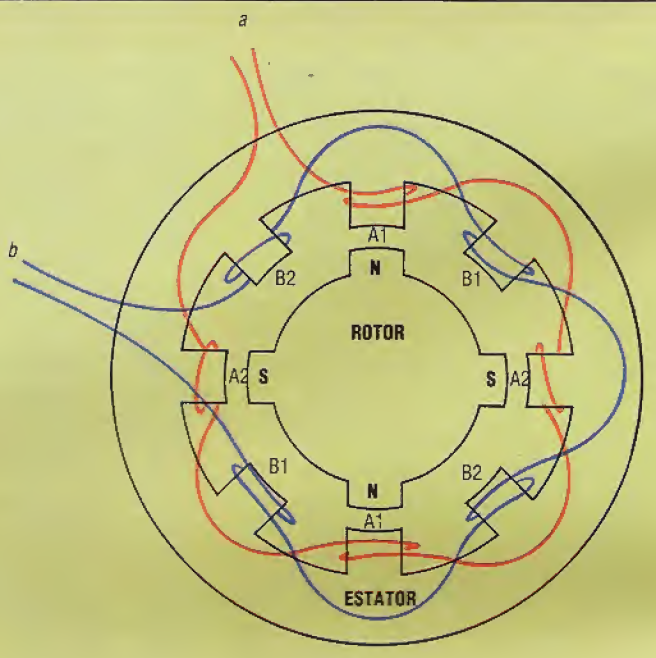
Cantidad	Artículo
1	Chip buffer 40109
3	Conector DIL de 16 patillas
2	Resistencia de 100 ohmios
2	Resistencia de 270 ohmios 0,5 W
2	Condensador de 0,1 μF
1	Condensador de 1000 μF 25 V
1	Veroboard de 24 franjas×50 agujeros
1	Rollo cable estañado 20 swg

PIEZAS DE RADIO

2	Activador de motor paso a paso SAA 1027

Un paso cada vez

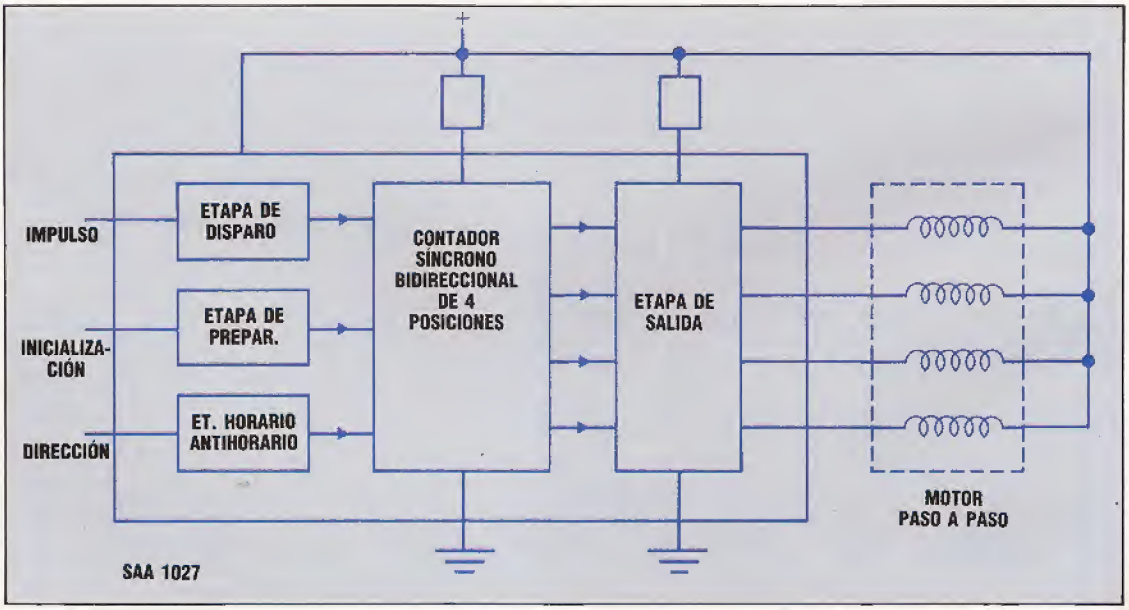
Este diagrama simplificado de un motor paso a paso muestra dos circuitos de bobinas del estator, *a* y *b*, y un rotor. El rotor se ve obligado a girar en sentido horario mediante la activación alterna de los dos circuitos de bobinas. Observe que los pares de bobinas A1 y A2 están bobinados en direcciones contrarias. Cuando se activa la bobina *a*, induciendo polos sur en el par A1, en el par A2 se inducen polos norte. Los pares de bobinas B1 y B2 están igualmente bobinados en sentidos contrarios. El ángulo mínimo de paso que puede describir este motor es de 45°. Los motores utilizados en nuestro robot poseen más bobinas en el estator y mayor número de polos en el rotor, lo que permite que roten en pasos de 7,5°



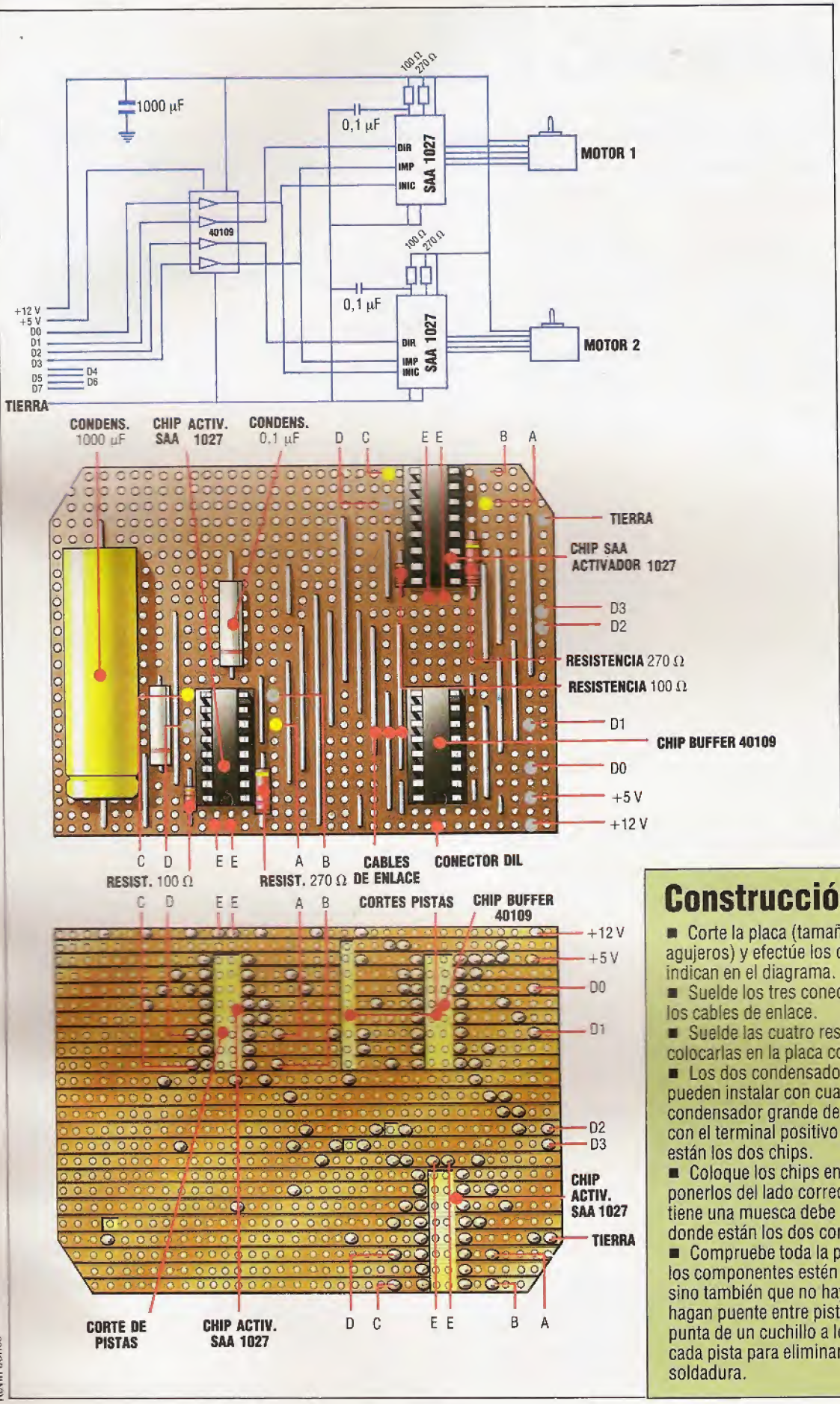
Kevin Jones

La fuerza motriz

Si bien la lógica de los chips activadores de motores paso a paso es compleja, los principios de operación son fáciles de comprender. Con el fin de hacer girar el rotor, se deben activar las bobinas del estator de acuerdo a una determinada secuencia. Un contador bidireccional se mueve a través de esta secuencia de una etapa cada vez en respuesta a una señal de impulso. Los pasos de la secuencia también se pueden efectuar en la dirección opuesta si se cambia la entrada de la línea de dirección, haciendo que el motor gire en el sentido contrario. Una tercera entrada permite, si es necesario, devolver el rotor a la posición que ocupaba al comienzo de la secuencia



Liz Dixon



Kevin Jones

El diagrama del circuito
La circuitería necesaria para activar los dos motores paso a paso es directa, utilizándose dos chips activadores SAA 1027 para proporcionar la adecuada secuencia de activación de bobinas para cada motor. Dado que los chips activadores operan a 12 V y las señales de la puerta para el usuario de su micro son de sólo 5 V, se emplea un chip buffer adicional para aislar la circuitería de 12 V del ordenador y traducir las señales de la puerta para el usuario, de menor voltaje, a las señales de mayor voltaje que requieren los chips activadores. En el próximo capítulo le mostraremos cómo se conecta la placa de circuitos con los motores y el enchufe D, y cómo efectuar las conexiones correctas con la puerta para el usuario del ordenador

Construcción de la placa

- Corte la placa (tamaño 24 franjas × 35 agujeros) y efectúe los cortes de pistas que se indican en el diagrama.
- Suelde los tres conectores de chips y después los cables de enlace.
- Suelde las cuatro resistencias. Es indistinto colocarlas en la placa con una u otra orientación.
- Los dos condensadores de 0,1 µF también se pueden instalar con cualquier orientación, pero el condensador grande de 1000 µF se debe instalar con el terminal positivo del lado de la placa donde están los dos chips.
- Coloque los chips en su sitio, asegurándose de ponerlos del lado correcto. El extremo del chip que tiene una muesca debe quedar del lado de la placa donde están los dos conectores.
- Compruebe toda la placa. Verifique no sólo que los componentes estén colocados correctamente, sino también que no haya gotas de soldadura que hagan puente entre pistas adyacentes. Pase la punta de un cuchillo a lo largo del agujero entre cada pista para eliminar cualquier resto de soldadura.

Atando cabos

Concluimos nuestro curso sobre el lenguaje assembly del 6809. Nos queda sólo atar algunos cabos sueltos del programa depurador

El módulo principal como primer objetivo establecer el mecanismo de interrupción, lo que nos permite colocar puntos de ruptura en el programa a depurar. Éstos transfieren el control al programa depurador y nos facilitan el poder inspeccionar el contenido de los registros y de las posiciones de memoria. Debemos seguidamente obtener la dirección de inicio del programa a depurar para que el control pueda pasar a ella por medio de la orden S. El resto de la rutina principal se refiere a la obtención de órdenes desde el teclado y su ejecución; el control se transfiere al programa a depurar por medio de las órdenes S y G y se devuelve al programa depurador por medio de las instrucciones SWI insertadas en los puntos de ruptura.

En la lección anterior ya se codificaron dos fases de inicialización para este módulo (véase p. 1297). El punto de entrada de las interrupciones sigue inmediatamente a la llamada de la subrutina. Aquí la primera instrucción S guardará (*Save*) el puntero de la pila pra que pueda ser usado para referenciar los valores de los registros guardados en la pila por la SWI. La siguiente fase es la interpretación de la orden. Ya hemos elaborado rutinas para producir todas las órdenes, con lo cual el problema se reduce aquí a seleccionar la subrutina apropiada para la orden entrada.

Uso de la tabla de salto

Es posible codificar esto con algo semejante a un conjunto de IF anidados, pero preferimos aprovechar la circunstancia de que la rutina Tomar-Orden devuelve un desplazamiento a la tabla de caracteres de órdenes y nos limitaremos a realizar estas llamadas empleando la tabla de salto. Quizá no sea éste el método más eficaz para el caso, pero es una técnica útil que vale la pena conocer. Requiere el establecimiento de una tabla de direcciones para cada una de las subrutinas que tratan una orden.

La instrucción JMP, a diferencia de las instrucciones de bifurcación, puede servirse de cualquiera de los modos normales de direccionamiento, incluidos el indexado y el indirecto. Si cargamos X con la dirección base de la tabla y empleamos el desplazamiento contenido en B (duplicado, ya que será una tabla de direcciones de 16 bits, no como la tabla de letras de órdenes que es de 8 bits), ocurrirá que la orden

```
JMP [B,X]
```

transferirá el control a la rutina apropiada. La llamada BSR se realiza a la dirección de esta instrucción de salto. Puesto que necesitamos establecer esta tabla por anticipado, es preciso añadir otra fase en la inicialización para llevar a cabo esta operación.

Proceso Est.-Tabla-Saltos

```
                es una tabla de 8 direcciones de 16 bits
                etc., son las direcciones de inicio de
las subrutinas de órdenes (commands)

FOR (para) cada subrutina
   Tomar dirección de inicio
   Guardar dirección de inicio en Tabla-Saltos
Endfor
```

Debemos ahora ocuparnos de lo que pasa al final de la ejecución, en el momento en que aparece la orden Q, abandonar (*quit*), aunque en realidad hay muy poco que hacer. Parece lógico dejar intactos tanto el depurador como el programa depurado, de modo que puedan ser ejecutados de nuevo si fuera necesario.

Al finalizar, la pila ha de quedar en la misma situación que tenía cuando comenzamos. Una solución pudiera ser el empleo de otra pila independiente para nuestro programa poniendo S a un valor nuevo y después restaurar el antiguo. Es una técnica generalmente útil, pero en nuestro caso puede resultar difícil encontrar un espacio libre en la memoria, teniendo el depurador colocado encima de otro programa. Una segunda solución consiste en incrementar sencillamente S en una cantidad adecuada para descartar lo que hayamos dejado allí, pero resulta difícil porque desconocemos si ha sucedido o no una interrupción y las cantidades en la pila serían diferentes. La solución más inmediata es guardar el valor inicial de S y restablecerlo como última operación del programa.

El mecanismo de interrupción, según se establece en el procedimiento de inicialización, almacena tres bytes en la dirección dada por el vector SWI en $FFFA; debemos almacenarla si no queremos obtener resultados desconcertantes en el momento en que el sistema operativo emplee SWI para sus operaciones. Necesitamos, pues, otra fase más de inicialización en la que guardemos estos valores que han de ser restaurados en nuestra rutina de abandono.

Proceso Guardar-Valores

```
Data:
   Guarda se compone de cinco bytes que almacenarán
   los valores a salvar
   Puntero-Pila es el valor en curso de S increm. en 2
   Vector-SWI se encuentra en $FFFA
Proceso:
   Guardar Puntero-Pila en Guarda
   Tomar Vector-SWI
   Guardar tres bytes de Vector-SWI en Guarda
```

La rutina de abandono (orden Q) no tiene más que invertir este proceso y devolver el control al sistema operativo. Lo cual puede realizarse de varias maneras: la instrucción SWI puede servir para ello, una vez restablecido su vector; o bien, se puede hacer un salto a un punto conocido de entrada en el sistema operativo. Es posible un salto a través del vector de reinicio en $FFFE, para devolver el control al sistema operativo, aunque esto puede provocar una inicialización de todo el sistema.

Proceso Abandono

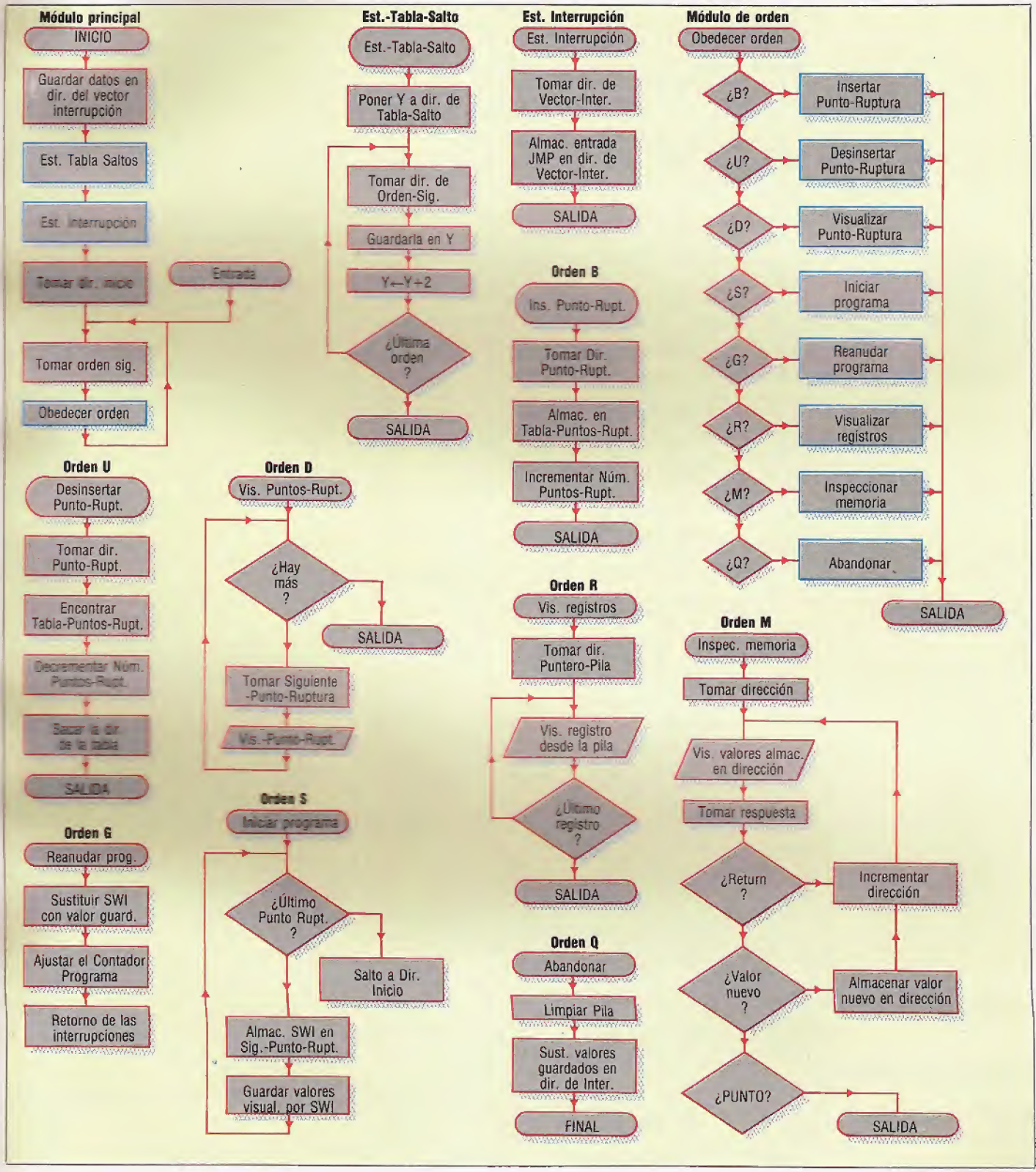
Data:

Guarda son cinco bytes para almacenar los valores a salvar
Puntero-Pila es el valor en curso de S, increm. en 2
Vector-SWI está en $FFFA
Vector-Reinicio está en $FFFE

Proceso:

Restaurar en Vector-SWI tres bytes de Guarda

Flujo del programa
Los presentes diagramas de flujo corresponden a los módulos del programa depurador de errores. Se han colocado según el orden en que son llamados por las restantes subrutinas. Dentro de los diagramas, las celdillas ribeteadas de azul indican rutinas individuales a las que se llama



Liz Dixon

Restaurar Puntero-Pila
Salto al sistema operativo

Ya estamos preparados para codificar el módulo principal. Muy poco ha cambiado el diseño desde que se esbozó en un principio, y esencialmente sigue siendo el mismo.

Módulo principal

Data:
Interrogación para entrada de la orden, es el carácter '>' en ASCII
Desplazamiento-Orden a la tabla de caracteres de órdenes y Tabla-Saltos
Proceso:
Guardar-Valores
Establecer-Tabla-Saltos
Establecer-Interrupciones
Tomar-Dirección-Inicio
REPEAT
Visualizar-Interrogación
Tomar-Orden
Ejecutar-Orden
INDEFINITELY

Con ello concluye nuestro programa depurador de errores. Parece que nos salió troceado en exceso, pero la fragmentación es típica de la programación modular. En este punto podríamos optimizar la codificación si nos dedicáramos a buscar atajos. Por ejemplo, se habrá dado cuenta de que hemos barajado una gran cantidad de valores sólo para cerciorarnos de que se encontraban en el registro adecuado para una subrutina; si ahora redefine el empleo de los registros, ciertamente ahorrará pasos reiterativos. Pero no se lo aconsejamos si el espacio de la memoria con que cuenta no es limitado. Según se van necesitando, hemos definido en bastantes sitios diferentes las mismas áreas de datos. Dos son las maneras de manejar las áreas de datos en el programa completo: se pueden retener los datos junto con el módulo que los emplea, que es la mejor opción teórica; o bien pueden definirse todos los datos juntos al comienzo del programa, lo cual tiene sus ventajas reales cuando se desea utilizar un desensamblador (o incluso un depurador) para el programa.

El depurador debe ser cargado en cualquier espacio libre de la memoria que el programa a depurar no ocupe o emplee. Se le da entrada mediante un salto al punto de entrada DEBUG (depurar); es, por tanto, necesario conocer su dirección antes de comenzar.

Establecer Tabla Salto

```
JTABLE  RMB   16          Espacio para ocho direcciones de dos bytes
SETUPJ  LEAY  JTABLE,PCR  Dir. base de tabla en Y
        LEAX  CMDB,PCR    Dirección inicio de la subrutina CMDB
        STX   ,Y++        La almacena en tabla
        LEAX  CMDU,PCR    Dirección inicio de subrutina CMDU
        STX   ,Y++        La almacena en tabla
        LEAX  CMDD,PCR    Dirección inicio de subrutina CMDD
        STX   ,Y++        La almacena en tabla
        LEAX  CMDS,PCR    Dirección inicio subrutina CMDS
        STX   ,Y++        La almacena en tabla
        LEAX  CMDG,PCR    Dirección inicio subrutina CMDG
        STX   ,Y++        La almacena en tabla
        LEAX  CMDR,PCR    Dirección inicio subrutina CMDR
        STX   ,Y++        La almacena en tabla
        LEAX  CMDM,PCR    Dirección inicio subrutina CMDM
        STX   ,Y++        La almacena en tabla
        LEAX  CMDQ,PCR    Dirección inicio subrutina CMDQ
        STX   ,Y++        La almacena en tabla
```

He aquí el salto efectivo a la subrutina. Asumimos que X contiene la dirección de JTABLE (Tabla Salto) y B del desplazamiento

```
DOCMD   JMP   [B,X]
```

Guardar Valores

```
SAVED   RMB   5           Cinco bytes por guardar
SAVEIT  LEAX  SAVED,PCR   Toma dir. para guardarla
        TFR   S,D         Transfiere S a D
        ADDD  #2          Suma dos en atención a la dirección de retorno
        STD   ,X++        Lo guarda
        LDY   $FFFA       Toma dir. de vector inter.
        LDA   ,Y+         Toma el primer byte a guardar
        STA   ,X+         Lo guarda
        LDD   ,Y          Toma otros dos bytes
        STD   ,X          Los guarda
        RTS
```

Orden Q

```
CMDQ    LEAX  SAVED,PCR   Dirección de Guarda
        LDY   $FFFA       Vector-SWI
        LDA   2,X         Primero de tres bytes
        STA   ,Y+         Restaurado
        LDD   3,X         Otros dos bytes
        STD   ,Y          Restaurados
        LDS   ,X          Puntero-Pila guardado
        JMP   [$FFFE]     Salto indir. por medio vector reinicio
```

Módulo principal

```
PROMPT  FCB   ,>          Puntero-Pila para Visualizar-Registros
STACKP  RMB   2
DEBUG   BSR   SAVEIT      Guardar-Valores
        BSR   SETUPJ      Establecer-Tabla-Salto
        BSR   INIT        Establecer-Interrupción y Tomar-Dirección-Inicio
ENTRY   STS   STACKP,PCR  Guardar Puntero-Pila
        LEAX  JTABLE,PCR
REPT02  LDA   PROMPT,PCR  Tomar Interrogación y visualizarlo
        BSR   OUTCH
        BSR   GETCOM      Tomar Orden
        LSLB              Doble despl. para tabla 16 bits
        BSR   DOCMD       Obedecer Orden
        BRA   REPT02      Siguiente Orden
```

Robots de élite

Fijemos nuestra atención en los robots más sofisticados que es posible encontrar en el mercado: los que se utilizan para enseñar los principios de la robótica y aquellos que son representativos del arte del diseño moderno de robots

Muchos de los robots que vamos a ver en este capítulo son caros, pero no entran en la categoría de robots industriales y están diseñados para su uso tanto en el hogar como en la escuela. El primer grupo que analizaremos es el que engloba los robots cuya ingeniería responde a un estándar elevado e incorporan muchas de las características de los brazos-robot industriales. La principal diferencia entre éstos y los brazos industriales es que la mayoría de los que mencionaremos aquí están diseñados para uso didáctico y se emplean para enseñar los principios de la robótica.

La principal diferencia entre estos brazos y sus equivalentes industriales radica en que éstos son más pequeños y tienen menos capacidad de manipulación de objetos grandes. En muchos casos, por supuesto, el mercado educativo se superpone al industrial, ya que si lo que la aplicación industrial necesita es un brazo relativamente ligero y pequeño, muchos de estos brazos cumplen con tal condición. Por ejemplo, puede que se requiera un robot industrial para manipular grandes lingotes de acero que pesen cientos de kilos, o para realizar una labor muy diferente, como es ensamblar componentes en una placa de circuito impreso, tarea ésta para la que no se requiere un brazo grande y potente. Por lo tanto, se puede considerar que los brazos-robot de esta categoría poseen el potencial para llevar a cabo aplicaciones serias, además de ser didácticos.

La segunda categoría que veremos es la de aquellos robots cuyo diseño incorpora los últimos adelantos logrados en robótica. Muchos de ellos estarán equipados con los poderes sensoriales que ya hemos analizado en anteriores capítulos.

Robots didácticos

Un brazo-robot de precio moderado es el Mentor, de Cybernetic Applications. Se vende en forma de modelo para armar, posee seis grados de libertad (cintura, hombro, codo y tres ejes de rotación en la muñeca) y está activado por motores eléctricos. Se lo puede controlar desde un BBC Micro, un Vic-20 o un Spectrum.

En amable compañía
Puesto que el Hero es a la vez móvil y posee una pinza, puede servirle a su dueño una taza de té por las mañanas (¡siempre y cuando, por supuesto, no haya que subir alguna escalera desde la cocina hasta el dormitorio!). En la fotografía vemos al Hero y al Mentor disponiéndose a saborear una taza de té

Ian McKinnell

La misma empresa produce, a un precio muy superior, el Neptune 1 y el Neptune 2. Estos brazos pueden levantar hasta 2,5 kg y funcionan con energía hidráulica, aunque utilizan agua y no el fluido hidráulico normal que emplean la mayor parte de los robots de esta clase. Estos brazos también se

Chris Stevens

Brazos para aprender

Las dos áreas más significativas de la investigación en microelectrónica son el diseño de chips y la robótica. Son pocos los aficionados que pueden experimentar en los talleres de sus hogares con nuevos microchips, pero los robots se están volviendo cada vez más accesibles. La finalidad principal de los brazos-robot reseñados en este capítulo no es otra que contribuir a una mayor comprensión del usuario acerca de cómo funcionan los robots. La mejor forma de acrecentar nuestros conocimientos sobre éstos es examinar sistemas de operación e intentar perfeccionarlos. Los brazos-robot que vemos en la fotografía, el Micro Grasp de Powertran Cybernetics y el Armdroid de Colne Robotics, se programa mediante el uso de microordenadores y poseen cinco grados de libertad. Se los puede adquirir totalmente montados o en forma de modelos para armar

NOMBRE	NEPTUNE 1	NEPTUNE 2	MENTOR	GENESIS P101	HERO
TIPO	Brazo	Brazo	Brazo	Brazo	Robot móvil
APLIC. PRINCIPAL	Didáctica	Didáctica	Didáctica	Didáctica	Didáctica/experimental
SENSORES	El potenciómetro registra la posición, la pinza puede detectar su grado de cierre	El potenciómetro registra la posición, la pinza puede detectar su grado de cierre	El potenciómetro registra la posición, la pinza puede detectar su grado de cierre	Realimentación posicional	Ultrasónico, permite la detección de movimiento. Los sensores detectan 256 niveles de luz y sonido. Sensores táctiles en la pinza
GRADOS DE LIBERTAD	6: cintura, hombro, codo y elevación de muñeca, tonel de muñeca y pinza	7: cintura, hombro, codo, elevación de muñeca, tonel de muñeca, guiñada de muñeca y pinza	6: cintura, hombro, codo y elevación de muñeca, tonel de muñeca y pinza	6: cintura, hombro, codo y elevación de muñeca, tonel de muñeca y pinza	4: hombro, codo, muñeca y pinza
PRECIO*	£1 250	£1 725	£345	£1 750	£1 995
ENERGIA	Hidráulica: bomba de agua activada por corriente eléctrica	Hidráulica: bomba de agua activada por corriente eléctrica	Eléctrica	Hidráulica: bomba de agua activada por corriente eléctrica	Baterías recargables
SE CONECTA AL	BBC Micro, Spectrum, Vic-20	BBC Micro, Spectrum, Vic-20	BBC Micro, Spectrum, Vic-20	Programado a través de caja controladora; interface estándar RS232; BBC Micro, Spectrum, Vic-20	Utilizando un ensamblador cruzado, el HERO se puede conectar a cualquier micro que posea una puerta en serie
FABRICADO POR	Cybernetic Applications Ltd., West Portway Industrial Estate, Andover, Hampshire SP10 3NN	Cybernetic Applications Ltd.	Cybernetic Applications Ltd.	Powertran Cybernetics, West Portway Industrial Estate, Andover, Hampshire SP10 3NN	Zenith Data Systems, Bristol Road, Gloucester, GL2 6EE

*En el mercado británico

pueden controlar con el BBC Micro, el Vic-20 y el Spectrum. El Neptune 2 tiene dos velocidades de operación diferentes; esto es de gran utilidad, porque se lo puede hacer mover rápidamente cuando se requieren grandes desplazamientos del brazo, y hacer luego que reduzca su velocidad para los movimientos que exijan mayor precisión.

Powertran Cybernetics produce el Genesis P101, que posee seis grados de libertad y se vende en forma de modelo para armar. Este modelo funciona con energía hidráulica y viene con una caja controladora para programar el robot, junto con una interface RS232 estándar gracias a la cual se lo puede conectar a la mayoría de los ordenadores. Este brazo también se puede adquirir premontado y probado a un precio superior.

Un interesante brazo de precio medio es el Cyber 310 de Cyber Robotics. Se vende premontado. Existen versiones para el BBC Micro, el Jupiter Ace, el Apple II, los Commodore Pet 3000/4000 y 8000 y el Hector HRX. Funciona mediante motores paso a paso y su capacidad para levantar pesos es de sólo 250 g, por lo cual es bastante ligero, pero la gama de opciones que ofrece es notable. Además de disponer de cinco grados de libertad, permite que el propio usuario controle la aceleración y deceleración del brazo, lo que significa que puede imitar la forma en la cual se mueve un brazo humano, alterando constantemente su velocidad según vaya cambiando la naturaleza de la tarea e imitando los efectos de la inercia. Todas las junturas se pueden mover simultáneamente y es posible especificar la posición del brazo ya sea como una posición relativa (p. ej., indicándole al brazo que se desplace *x* unidades hacia adelante respecto a su posición actual) o bien como una posición absoluta (especificando un movimiento hasta algún punto en relación a una posición "base"). Puede ser programado en BASIC y también en una versión de FORTH, conocida como Robo FORTH y desarrollada por la firma fabricante.

Ascendiendo en la gama de precios llegamos al HRA 933 y HRA934 de Feedback Instruments, que se venden ya montados. Ambos son brazos que funcionan con energía hidráulica y pueden levantar 1,35 kg con una precisión de posicionamiento de 3 mm. Además de contar con sensores de posición para las junturas del brazo, los brazos poseen sensores táctiles en sus efectores finales. Estos sensores indican que han cogido algo y permiten que el brazo controle la fuerza aplicada en el momento de recoger objetos. El control se realiza a través de una interface RS232 y las instrucciones específicas para el control se imparten utilizando el Apple II, el Tandy TRS-80, el Commodore PET, el AIM 65 y el MAT385.

CYBER 310	HRA933	HRA934
Brazo	Brazo	Brazo
Didáctica	Didáctica	Didáctica
Pinza	Capacidad para determinar su posición, la pinza puede detectar su grado de cierre	Capacidad para determinar su posición, la pinza puede detectar su grado de cierre
5 rotación de base, codo, rotación de muñeca, articulación de muñeca y pinza	5 rotación de base, codo, rotación de muñeca, articulación de muñeca y pinza	5 rotación de base, codo, rotación de muñeca, articulación de muñeca y pinza
£ 650	£ 2 195	£ 2 726
Eléctrica	Eléctrica	Eléctrica
BBC Micro, Jupiter Ace, Apple II, serie Commodore PET, Hector HRX	Apple II, Tandy TRS80, serie Commodore PET, AIM 65, BBC Micro, MAT 385	Apple II, Tandy TRS80, serie Commodore PET, AIM 65, BBC Micro, MAT 385
Cyber Robotics, 61 Ditton Walk, Cambridge, CB5 8QD	Feedback Systems Ltd., Park Road, Crowborough, East Sussex, TN6 2QR	Feedback Systems Ltd.

La robótica en estos momentos

El robot Hero-1 de Zenith Data Systems se vende en forma de modelo para armar y ofrece algunas facilidades bastante notables. Es móvil y posee un brazo que hace uso de un sistema de coordenadas esféricas en virtud del cual el brazo puede extenderse y contraerse telescópicamente. El Hero está equipado con una gran matriz de sensores para detectar movimiento, sonido y luz, incluyendo un sensor ultrasónico de distancia que contribuye a evitar colisiones, y un sintetizador de voz que le proporciona un vocabulario ilimitado. Asimismo, posee un brazo con cinco grados de libertad. El ensamblaje es muy laborioso; por este motivo es posible que el interesado desee adquirirlo ya armado. En este caso el precio se eleva sustancialmente.

Todos estos robots son, en cierto sentido, poco más que un entretenimiento de lujo, desde el punto de vista de lo que realmente pueden hacer para el usuario; y, en realidad, hasta el momento su principal uso ha sido por parte de firmas comerciales que desean emplear un robot con fines publicitarios: repartir en mano folletos en los puestos de las ferias de muestras o hacer demostraciones de productos. No obstante, representan la mejor tecnología de la robótica existente en estos momentos. Todos ellos utilizan sensores de una forma inteligente, se desplazan inteligentemente y poseen brazos inteligentes. Ninguno de ellos dispone de algún sistema de visión, pero sí pueden hablar y pueden oír señales acústicas y responder a las mismas.

Huelga decir que su precio disuadirá a muchas personas de la tentación de comprarlos, pero no por ello dejan de estar allí, como algo a lo que aspirar, algo que quizá usted mismo sea capaz de hacer realidad construyéndose su propio robot. Asimismo, constituyen un indicio del ritmo al cual está avanzando la robótica.

Despegue vertical

Iniciamos una serie en que analizaremos el denominado "software vertical", desarrollado para cumplir una función específica relacionada con medicina, derecho, periodismo, fotografía y otras actividades especializadas

Existen muchísimas aplicaciones para los ordenadores personales, muchas de las cuales no son evidentes. Estas aplicaciones se han desarrollado a consecuencia de la adaptación creativa de software ya existente o de la generación de software para uso especial. Se dice que éstas son aplicaciones del "mercado vertical" porque se aplican a un grupo específico de personas, como pueden ser médicos, químicos o psicólogos. En esta serie analizaremos numerosos paquetes para el mercado vertical que revelan nuevas e interesantes facetas del uso de los microordenadores. Los siguientes ejemplos ayudarán a ilustrar la clase de problemas para cuya solución se diseña software vertical.

Un grupo de padres de adolescentes adictos a la heroína, residentes en el West End londinense, está utilizando el programa *BrainStorm*, de Caxton Software, para planificar una campaña de divulgación de sus actividades entre los médicos, asistentes sociales y órganos para la aplicación de la ley de la localidad. El dueño de un restaurante de Kent está utilizando una hoja electrónica *Practicalc* para analizar los pedidos de los clientes, con el fin de planificar sus futuros menús. Una granja de Sussex está utilizando el mismo programa (trabajando con cuatro Commodore 64) para manejar todo, desde sus proyectos de gran importancia hasta el control de sus costos de energía. Un cirujano de un hospital de Londres está aplicando el programa *Superbase*, de Precision Software, en la investigación que está llevando a cabo sobre las causas del cáncer y su curación.

En Gran Bretaña hay médicos que están empleando ordenadores para cumplir con la nueva normativa de que todas las etiquetas de las recetas de los pacientes deben ir impresas y no escritas a mano. Un diseñador de cocinas, de Lancashire (Gran Bretaña), utiliza un programa (desarrollado para un BBC Micro por una pareja de amigos con los que juega al billar) para ir colocando hornos, neveras y otros artefactos de cocina en diferentes sitios sobre un plano del espacio disponible basado en pantalla. De hecho, le ha resultado una ayuda tan eficaz en su negocio de muebles de cocina (y dormitorio), que ha formado, junto con los diseñadores originales del programa, una sociedad para ofrecerles el sistema a otros comerciantes.

Éstos son apenas unos pocos ejemplos de las nuevas respuestas a la ya tradicional pregunta que formula todo probable comprador de un ordenador nuevo: "¿Para qué sirve?" Se ha calculado que el usuario medio de un ordenador no explota más del 10 % del potencial de la máquina, y ésta puede ser una estimación incluso optimista. Invertir dinero en algo que luego se limita en un uso final, ya sea entretenerse con juegos o administrar las cuentas, no es tan rentable, teniendo en cuenta el costo, como explotar su versatilidad al máximo.

En este sentido existen dos alternativas. Una opción es encontrar nuevas aplicaciones para el software estándar. Un fotógrafo profesional de Sheffield utiliza el módulo de control de stocks del paquete *Anagram integrated accounts* para administrar su biblioteca de fotografías, con lo que consigue sacar rendimiento de los originales de su archivo de copias y transparencias antiguas. Del mismo modo, algunas agencias de colocaciones utilizan la base de datos *Tomorrow's office* para cotejar las necesidades de sus clientes con el potencial humano disponible, conservando los datos de los *curriculum vitae* en disco rígido y enviándolos por correo automáticamente. Esto es mucho más barato que utilizar paquetes diseñados específicamente para esa tarea: el paquete *Body matching and marketing*, producido por la firma AP Computer Consultants, tiene un precio que sobrepasa las mil libras esterlinas (unas 200 000 ptas.).

Diseño a medida

La alternativa es buscar un paquete diseñado para el uso específico que uno necesite. Imagínese que es un estudiante de teología y tiene la mesa de estudio abarrotada de tomos enormes, concordancias, notas, diccionarios bíblicos y cosas por el estilo. Bueno, puede que lo que necesite sea el *"The Word" processor* (procesador de la Palabra), de Bible Research Systems para el IBM y máquinas compatibles. Este paquete incluye la traducción completa del rey Jacobo I, con facilidades completas de búsqueda para la creación de referencias cruzadas en disco.

Y si no se fía de ninguna de las traducciones del material bíblico, puede compararlas con el original mediante un programa llamado *The Greek transliterator*, que le dará el equivalente en griego de cualquier palabra o frase en inglés, y visualizará todos los casos en los que aparezca en el texto en inglés. Esto permite comparar las diversas formas en que se ha traducido una palabra. Estos estudios electrónicos de la Biblia, sin embargo, no son baratos. Estos dos programas le supondrán un gasto que ascenderá a las 253 libras esterlinas cada uno (unas 50 000 ptas.).

Vamos a ver otro paquete que a primera vista parecería igualmente improbable de hallar. Si usted es un ingeniero que está planificando desagües y alcantarillas, entonces el *MIDDUSS* (el *McMaster interactive design of stormwater systems*) le ayudará a medir tuberías, canales y estanques de almacenamiento, permitirá que genere hidrográficos y le proporcionará vuelcos en pantalla de todo su tra-

bajo en gráficos en alta resolución. Para ejecutar todo esto necesitará un Sirius de 256 K.

Los sistemas de alcantarillado parecen haber proporcionado a los programadores muchísimo estímulo. El robusto ordenador de mano Husky es el corazón del programa *CAMIL (Computer-aided manhole inspection and location)*, desarrollado en Southampton y que ahora están aplicando en toda Gran Bretaña las autoridades de aguas. El programa permite que los inspectores entren datos en el campo y los mismos sean transmitidos a través de líneas telefónicas a los ordenadores centrales, que pueden entonces imprimir trazados de las cloacas. El programa puede incluso responder a una petición para ver "todas las cloacas de ladrillo construidas antes de 1900"; y de éstas hay una cantidad abrumadora.

A los viajantes de comercio les resultará sumamente útil la serie de programas "Travelling" que se ejecutan en otro ordenador de mano: el excelente NEC PC-8201. Además de paquetes básicos como el *Travelling writer* (escritor viajante: un procesador de textos para redactar informes, con capacidades para correo electrónico y administración de datos), la serie incluye el *Travelling project manager*, el *Travelling appointment manager*, el *Travelling sales manager* y (el más esencial) el *Travelling expense manager* (administrador de proyectos, de citas, de ventas y de gastos, respectivamente).

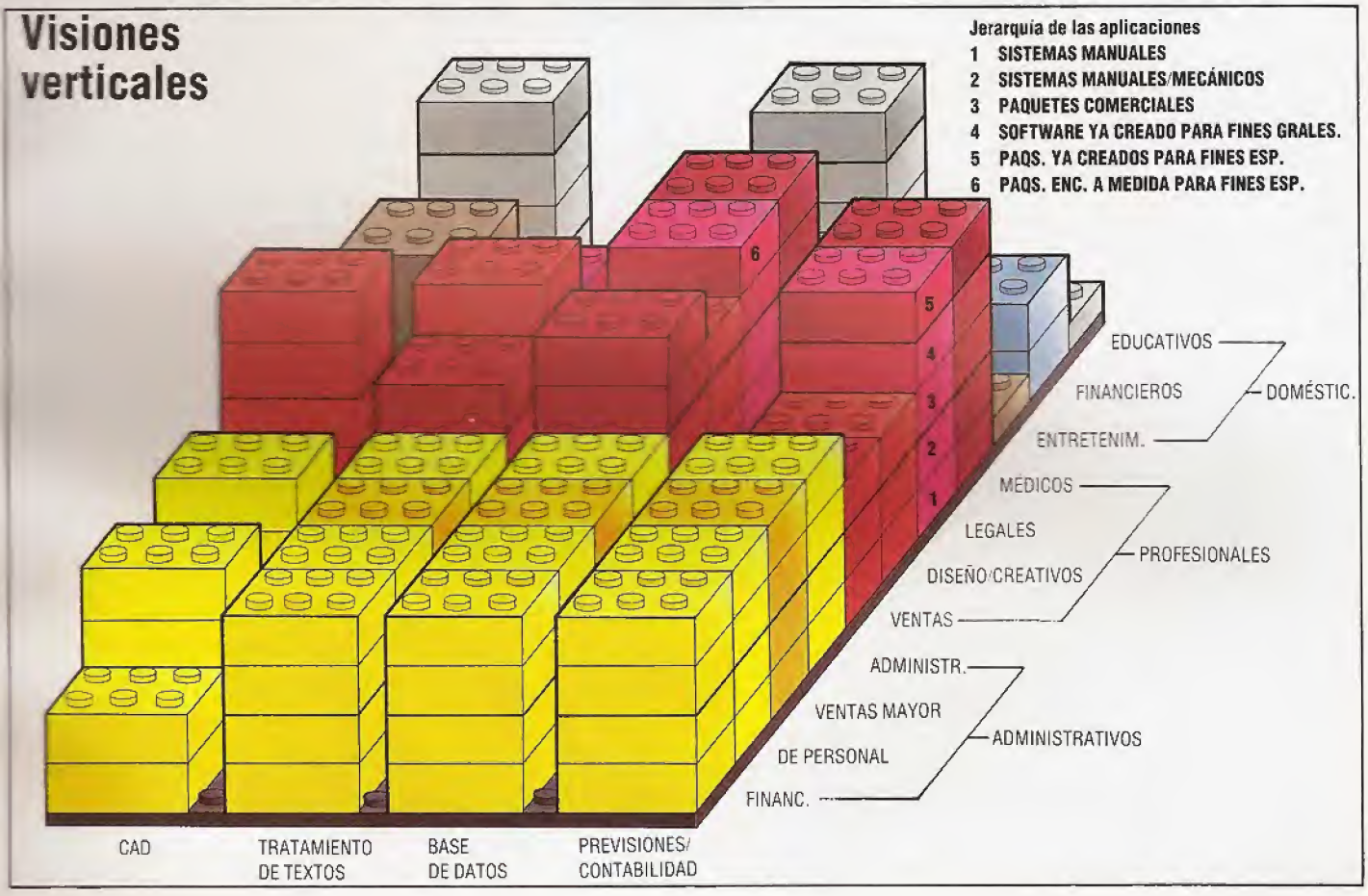
Los vendedores también pueden mejorar sus técnicas de venta con el módulo Sales Edge del juego de programas *Human edge*. Éstos son distribuidos por Thorn EMI para el IBM y el Apricot. El módulo Sales Edge evalúa los puntos fuertes y los puntos débiles del usuario de cara a la venta mediante una serie de preguntas y respuestas del tipo acuerdo/desacuerdo; tras una serie similar de preguntas al cliente, sugiere una estrategia de venta con maniobras de "apertura" y "cierre".

Si piensa invertir en divisas, quizá sienta la tentación de adquirir el Forextend de Forexia, que le permite estudiar y analizar lo que está sucediendo con el dólar, la libra esterlina, el franco suizo, el yen japonés y el marco alemán. El programa produce 37 gráficas de comparaciones, indicadores relativos de robustez, tasas de interés e índices de influencia comercial para cada día del período entre el 1 de octubre de 1983 y el día actual.

Los programas de citas y horarios están, asimismo, adquiriendo gran popularidad. La mayoría de los ordenadores tienen relojes incorporados, pero pocos de ellos pueden interrumpir cualquier actividad que el usuario esté desarrollando para recordarle dónde debería estar en ese momento. No obstante, el planificador de horarios basado en ROM de Hewlett-Packard para su HP-75C hace exactamente eso y, además, con toda una variedad de sonidos de aviso diferentes.

En el próximo capítulo de esta serie analizaremos en profundidad el *BrainStorm*, un programa del cual se afirma que organiza sus pensamientos de la misma manera en que un procesador de textos organiza sus oraciones. Y, por consiguiente, analizaremos paquetes de software para personas dedicadas al comercio, la medicina, la educación, la investigación, el periodismo, el derecho, el video, los espectáculos y la publicidad, con casos concretos de usuarios específicos.

Perfiles de mercado
El software se escribe para satisfacer las necesidades de grupos de usuarios primarios y para venderlo en los sectores fundamentales del mercado. El grueso del mismo, por lo tanto, comprende en gran parte paquetes financieros, de tratamiento de textos y de base de datos; una cuarta área, la del diseño asistido por ordenador (CAD: *Computer-aided design*), está creciendo en importancia. En el diagrama, estas áreas de aplicaciones se yuxtaponen a una selección de usuarios. El eje vertical representa el nivel de complejidad de la aplicación, estando los métodos manuales en el nivel inferior, el software para ordenador de uso general en el nivel medio, y el software diseñado a medida o con fines específicos en el nivel superior. Los perfiles de uso resultantes muestran que el software para fines generales suele adaptarse bien a las exigencias empresariales, pero que los usuarios necesitan adaptar a su medida los paquetes o bien crearse los suyos; los perfiles alto-bajo irregulares indican una descompensación entre las necesidades y los recursos, los perfiles de altura media regulares muestran la capacidad del mercado para equilibrar necesidades y clientes



Tomar y llevar

Prosiguiendo con nuestro juego, desarrollaremos las rutinas necesarias para recoger y transportar objetos entre escenarios

En el último capítulo de este proyecto vimos el análisis de las instrucciones y diseñamos un grupo de instrucciones "normales". En este grupo incluimos las instrucciones RECOGER y DEJAR, junto con sus variantes COGER y PONER. Una vez reconocida la instrucción apropiada, podemos construir las rutinas que obedezcan la instrucción. En primer lugar vamos a considerar RECOGER.

Para entender los métodos que emplea la rutina RECOGER, resumamos la forma en que el programa lleva el registro de los objetos de nuestro mundo de aventuras. En el primer capítulo del proyecto diseñamos sentencias DATA para cada escenario que contenían descripciones del mismo, los nombres de los objetos presentes e información sobre las salidas posibles. Después de leer los datos, la matriz IVS(,) (utilizada para almacenar los datos de los objetos de *El bosque encantado*) posee el siguiente contenido:

N	IVS(N,1)	IVS(N,2)
1	ESCOPETA	10
2	FAROL	9
3	LLAVE	5

La primera columna de la matriz contiene el nombre del objeto, mientras que la segunda contiene el número de su escenario inicial en el mapa del mundo de aventuras. Durante la descripción de cualquier escenario, se explora la segunda columna de esta matriz para ver si cualquiera de los objetos se halla en el escenario actual del jugador, P. Cuando éste desea recoger un objeto de un escenario, utilizando una instrucción ajustada al formato RECOGER EL OBJETO, se deben considerar varios factores:

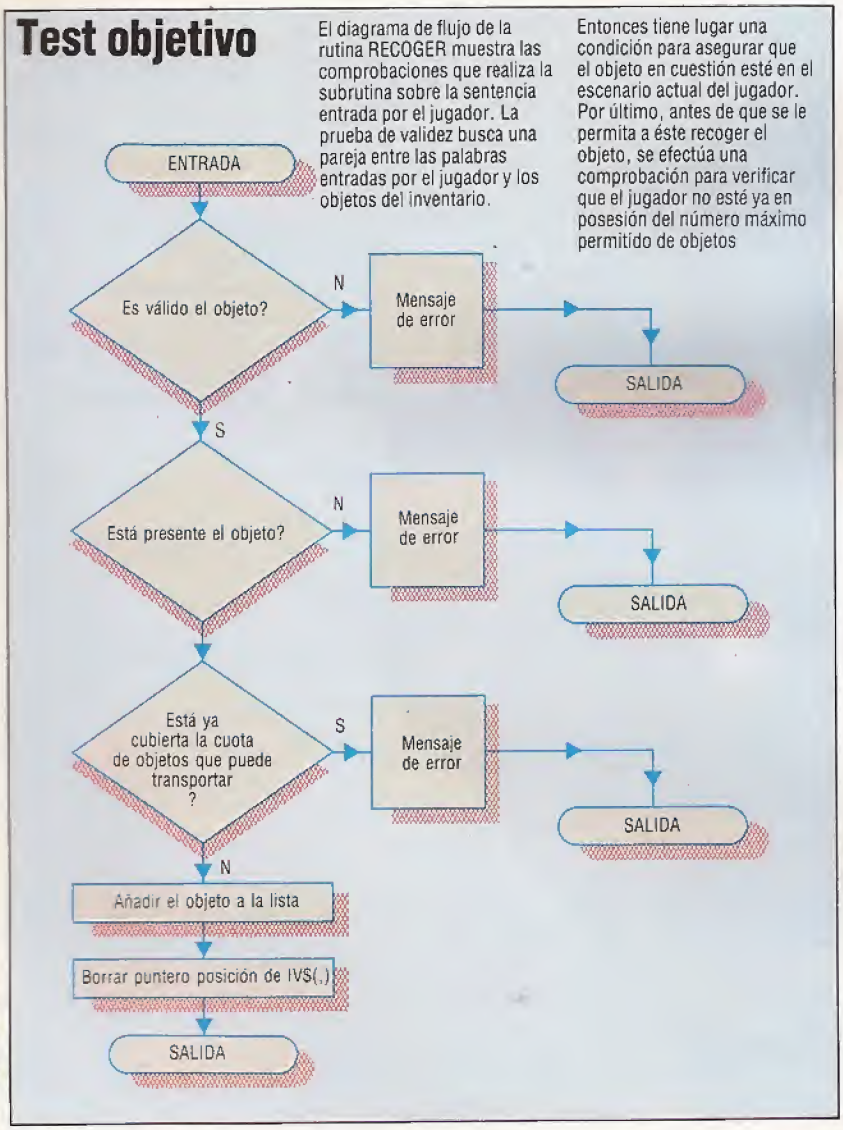
- ¿Es válido el objeto de la instrucción?; en otras palabras, ¿aparece en la matriz del inventario, IVS(,)?
- ¿Está presente el objeto en el escenario actual del jugador?
- ¿Posee ya éste la cuota completa de objetos que permiten las reglas del juego?

Si se puede responder satisfactoriamente a estas preguntas, entonces el jugador puede recoger el objeto. Esto implica añadir la descripción del objeto a la matriz de objetos personales del jugador, ICS(), y borrar el indicador de posición de la entrada correspondiente en IVS(,). Observe que el nombre del objeto no se tiene que borrar. Si utilizamos un indicador de posición de −1 para cada objeto que se haya recogido y transportado, tales objetos no aparecerán en las descripciones de los escenarios. Sería muy curioso coger la ESCOPETA en el escenario 10, desplazarse hasta el escenario 9 y después otra vez hasta el 10, para encontrarse al regreso con que la ESCOPETA todavía está allí. Por lo tanto, la matriz IVS(,) lleva un registro de las posiciones de todos los objetos que el jugador *no* está llevando consigo. El diagrama de flujo para la rutina RECOGER muestra la sencilla lógica que se debe aplicar.

Test objetivo

El diagrama de flujo de la rutina RECOGER muestra las comprobaciones que realiza la subrutina sobre la sentencia entrada por el jugador. La prueba de validez busca una pareja entre las palabras entradas por el jugador y los objetos del inventario.

Entonces tiene lugar una condición para asegurar que el objeto en cuestión esté en el escenario actual del jugador. Por último, antes de que se le permita a éste recoger el objeto, se efectúa una comprobación para verificar que el jugador no esté ya en posesión del número máximo permitido de objetos




```
3700 REM **** S/R RECOGER ****
3710 GOSUB 5300:REM ES VALIDO EL OBJETO
3720 IF F=0 THEN SN$="NO HAY NINGUN  "+WS:GOSUB5500:
     RETURN
3730 OV=F:GOSUB5450:REM COMPROBAR INVENTARIO
3740 IF HF=1 THEN SN$="TU YA LLEVAS  "+IVS(F,1):GOSUB 5500:
     RETURN
3750 :
3755 REM ** ESTA AQUI EL OBJETO  ? **
3760 IF VAL(IVS(F,2))<>P THEN SN$=IVS(F,1)+"  NO ESTA
     AQUI":GOSUB5500:RETURN
3770 :
3780 REM ** AÑADIR OBJETO A LA LISTA **
3790 A=0
3800 FOR J=1 TO 2
```

Gary Capps-Jenner

```
3810 IF ICS(J)="" THEN ICS(J)=IV$(F,1):AF=1:J=2
3820 NEXT J
3830 '
3840 REM ** CUOTA COMPLETA **
3850 IF AF=0 THEN PRINT"YA TIENES DOS OBJETOS": RETURN
3860 '
3870 SN$="RECOGES EL "+IV$(F,1):GOSUB5500
3880 IV$(F,2)="-1":REM BORRAR ENTRADA INVENTARIO
3890 RETURN
```

Ahora vamos a analizar cada una de estas tres condiciones por separado.

La prueba de validez

De las tres comprobaciones, la más complicada e importante es la *prueba de validez*. En su forma más simple, ésta podría ser una rutina que se limitara a tomar la segunda parte de la instrucción descompuesta y compararla con cada uno de los elementos de la matriz del inventario, IV$(,). Sin embargo, si fuera éste el caso, la instrucción RECOGER se vería limitada a la estructura rigurosa de RECOGER OBJETO. Incluso variaciones tales como RECOGER LA ESCOPETA serían inaceptables, puesto que la rutina intentaría emparejar LA ESCOPETA con el inventario, en vez de emparejar sólo ESCOPETA. Para permitir cierta flexibilidad en la estructura de la instrucción RECOGER hemos de desarrollar un método más sofisticado para comparar la segunda parte de la instrucción dada con el inventario de objetos.

El procedimiento más evidente para aumentar la flexibilidad consiste en dividir la segunda parte de la instrucción dada en las palabras que la componen y luego comparar cada una de ellas con el inventario de objetos. Si bien con ello solucionaríamos el problema esbozado anteriormente, este método también tiene sus inconvenientes. Si, por ejemplo, quisiéramos utilizar una descripción de dos palabras para un objeto, como CUCHILLO GRANDE, entonces utilizando este método la instrucción RECOGER EL CUCHILLO GRANDE no tendría pareja. La rutina compararía las palabras EL, CUCHILLO y GRANDE por separado con la lista del inventario. Este problema se puede solventar haciendo que la rutina sea aún más refinada. En vez de buscar una pareja exacta, se podría diseñar una rutina que explorara cada descripción de objetos del inventario para la palabra de instrucción que se esté examinando, desplazándose letra por letra a través del nombre del objeto hasta encontrar una pareja o hasta llegar al final del nombre del objeto. La pantalla muestra cómo se realiza esto.

Una ventaja que ofrece el buscar una pareja entre instrucción e inventario de esta manera, es que se pueden dar en la instrucción versiones abreviadas de la palabra objeto. En el ejemplo de arriba, la instrucción RECOGER EL CUCH también formaría la pareja correcta, suponiendo que antes de CUCHILLO GRANDE no hubiera en el inventario ningún otro nombre de objeto que tuviera la combinación de letras CUCH. De ser así, entonces se formaría una pareja incorrecta con la entrada anterior del inventario. Los problemas de este tipo forman parte del precio que se debe pagar por la mayor flexibilidad de la rutina. La mayoría de los problemas de emparejamiento incorrecto se pueden eliminar mediante una cuidadosa selección de los nombres de los objetos. Si dos nombres de objeto deben contener el mismo grupo de caracteres, o si un nombre es una subserie de otro (como SOL y

La gran coincidencia


```
  10 REM **** DEMOSTRACION EMPAREJAR UN OBJETO ****
  40 MODE 5:COLOUR 2:DIM V$(3)
  60 FOR I=1 TO 3:READ V$(I):NEXT I
  90 A$="THE":B$="KNIFE":C$="KNI"
  95 S$="                "
 110 M$=A$:GOSUB 1000:REM EMPAREJAR "THE"
 130 M$=B$:GOSUB 1000:REM EMPAREJAR "KNIFE"
 150 M$=C$:GOSUB 1000:REM EMPAREJAR "KNI"
 160 END
1000 REM **** S/R EMPAREJAR ****
1010 CLS:F=0:LW=LEN(M$)
1030 FOR J=1 TO 3:LI=LEN(V$(J))
1042 X=1:Y=6:GOSUB 2000:PRINT S$
1045 X=1:Y=6:GOSUB 2000:PRINT V$(J)
1047 FOR I=1 TO LI-LW+1
1050 X=1:Y=5:GOSUB 2000:PRINT S$
1060 X=I:Y=5:GOSUB 2000:PRINT M$
1070 IF MID$(V$(J),I,LW)=M$ THEN F=1:I=LI:J=3
1075 FOR D=1 TO 300:NEXT D,I:REM DEMORA
1080 IF F<>0 THEN GOSUB 2500
1085 NEXT J:RETURN
2000 REM **** POSICIONAR CURSOR EN X,Y ****
2010 PRINT TAB(X,Y);:RETURN
2500 REM **** HALLADA PAREJA ****
2510 COLOUR 1:X=F:Y=6:GOSUB 2000:PRINT M$
2511 FOR K=1 TO 5:X=1:Y=6:GOSUB 2000:PRINT S$
2512 FOR D=1 TO 500:NEXT D,K:REM DEMORA
2520 A$=GET$:COLOUR 2:RETURN
3000 REM **** DATOS INVENTARIO ****
3005 DATA "SMALL FORK","RED DOOR","LARGE KNIFE"
```

```
  10 REM ** PAREJA SPECTRUM **
  40 INK 6:DIM V$(3,20)
1070 IF V$(J,I TO I+LW-1)=M$ THEN F=I:I=LI:J=3
2010 PRINT AT (Y,X);:RETURN
2502 INK 2:X=F:Y=6:GOSUB2000:PRINT S$
2520 A$=INKEY$:IF A$="" THEN 2520
2525 INK 6:RETURN
```

```
  10 REM ** PAREJA CBM 64 **
  40 PRINT CHR$(158):DIM V$(3)
  50 DN$=CHR$(17):FOR K=1 TO
     5:DN$=DN$+DN$:NEXT:DN$=CHR$(19)+DN$
2010 PRINT LEFT$(DN$,Y)TAB(X);:RETURN
2502 PRINTCHR$(28):X=F:Y=6:GOSUB2000:PRINT S$
2520 GETA$:IF A$="" THEN 2520
2525 PRINT CHR$(158):RETURN
```

La subrutina de prueba de validez diseñada para utilizar junto con la rutina RECOGER explora la sentencia entrada de palabra en palabra, tratando de hallar una pareja con una entrada del inventario. Este breve programa ilustra la forma en que la rutina de validez va buscando una pareja a lo largo del inventario. Para esta demostración, en el inventario hay tres objetos, y el programa intenta emparejar las palabras "THE", "KNIFE" y "KNI". Cada vez que se encuentra una pareja, el programa espera que se pulse una tecla antes de seguir adelante

SOLDADO), entonces el más corto de los dos nombres se debe colocar en un lugar anterior en la matriz del inventario. Además, no se deben utilizar descripciones de objetos diferentes que contengan las mismas palabras, como CUCHILLO GRANDE.

```
5300 REM **** S/R OBJETO VALIDO ****
5310 NN$=NN$+" ":LN=LEN(NN$):C=1:F=0
5315 FOR K=1 TO LN
5320 IF MID$(NN$,K,1)<>" " THEN NEXT K:RETURN
5325 W$=MID$(NN$,C,K-C):C=K+1
5330 LW=LEN(W$)
5335 FOR J=1 TO 3
5340 LI=LEN(IV$(J,1)):REM LONGITUD DEL OBJETO
5350 FOR I=1 TO LI-LW+1
5360 IF MID$(IV$(J,1),I,LW)=W$ THEN F=J:I=LI:J=3:K=LN
5370 NEXT I,J,K
5380 RETURN
```

Habiendo hallado una pareja, la rutina establece la variable F como el elemento de la matriz del inventario que corresponde al objeto de la instrucción. De no hallarse pareja, el valor de F permanece en 0, indicando que tal objeto no existe en el juego.

¿Está presente el objeto?

Una vez se ha establecido el número de matriz del objeto mediante la subrutina de prueba de validez, el escenario del objeto se puede cotejar fácilmente con la variable de escenario actual, P. El objeto a recoger está en IV$(F,1) y su situación está en IV$(F,2). La línea 3760 de la rutina RECOGER de *El bosque encantado* compara este valor con el de P. No obstante, el mensaje de error generado ("el OBJETO no está allí") podría no ser estrictamente correcto. El objeto podría estar presente en el escenario actual, pero en manos del jugador. Por consiguiente, se debe efectuar una comprobación para ver si éste transporta consigo el objeto en cuestión antes de generar el mensaje de error. De ser así, se puede producir un mensaje de error diferente (como "Tú ya tienes el OBJETO"). La siguiente subrutina verifica el inventario principal y establece un indicador, HF, a uno si el objeto lo lleva consigo el jugador. Esta condición se indica mediante un −1 en el elemento correspondiente de la matriz.

```
5450 REM **** S/R TIENE EL JUGADOR EL OBJETO ****
5460 HF=0
5470 IF IV$(OV,2)="-1" THEN HF=1
5480 RETURN
```

Comprobación de la lista

Estas dos tareas de verificar si la lista del jugador está completa y añadir a la lista se pueden combinar. Utilizando la matriz IC$() para retener los objetos transportados, se puede emplear un bucle FOR...NEXT para localizar el primer espacio libre de la matriz, de modo que se pueda entrar el nuevo objeto. Según las reglas de *El bosque encantado*, el jugador sólo puede transportar dos objetos en cualquier momento dado. Por lo tanto, el bucle FOR...NEXT utilizado sólo se ejecuta dos veces. De hallarse un espacio libre, entonces se entra el nuevo objeto; si no, sale en la pantalla un mensaje indicando que el jugador ya lleva consigo dos objetos.

La tarea final consiste en borrar el puntero de posición del objeto recién recogido del inventario. Esto se realiza poniendo IV$(F,2) a −1.

Ahora que el jugador tiene capacidad para recoger objetos, podemos incluir otra instrucción. Suele ser útil que el jugador pueda ver qué objetos está transportando. Por ejemplo, si se encuentra con una puerta cerrada con llave, puede haber olvidado que 20 movimientos antes recogió una llave. Permitir que el jugador liste los objetos que lleva consigo es una ayuda útil para la memoria. La codificación requerida es simple: se emplea un bucle FOR...NEXT para visualizar el contenido del inventario de objetos del jugador, IC$().

```
4100 REM **** LISTAR INVENTARIO TRANSPORTADO ****
4110 PRINT"OBJETOS QUE LLEVAS CONTIGO:"
4120 FOR I=1 TO 2
4130 PRINT" ";IC$(I)
4140 NEXT I
4150 RETURN
```

Listados Digitaya

```
2140 REM **** S/R RECOGER ****
2145 IV$(4,1)="BILLETE AL TRIESTADO"
2150 GOSUB5730:REM ES VALIDO EL OBJETO
2160 IF F=0 THENPRINT"NO HAY NINGUN ":W$:RETURN
2170 REM ** YA SE HA RECOGIDO EL OBJETO? **
2180 OV=F:GOSUB5830
2190 IFHF=1 THEN SN$="TU YA TIENES
     EL " +IV$(F,1):GOSUB5880:RETURN
2200 :
2210 REM ** ESTA AQUI EL OBJETO **
2220 IF VAL(IV$(F,2))<>P THEN SN$=IV$(F,1)+"NO ESTA
     AQUI":GOSUB 5880:RETURN
2230 :
2240 REM ** AÑADIR OBJETO A LA LISTA **
2250 AF=0:FOR J=1TO4
2260 IF IC$(J)="THENIC$(J)=IV$(F,1):AF=1:J=4
2270 NEXT J
2280 :
2290 REM ** COMPROBAR SI CUOTA CUBIERTA **
2300 IF AF=0 THENPRINT "YA LLEVAS CUATRO OBJETOS":
     RETURN
2310 :
2320 SN$="RECOGES EL "+IV$(F,1):GOSUB5880
2330 IV$(F,2)="-1":REM BORRAR ENTRADA POSICION
2340 RETURN

5730 REM **** S/R OBJETO VALIDO ****
5740 NN$=NN$+" ":LN=LEN(NN$):F=0:C=1
5745 FOR K=1 TO LN
5750 IF MID$(NN$,K,1)<>" " THEN NEXTK:RETURN
5755 W$=MID$(NN$,C,K-C):C=K+1:LW=LEN(W$)
5760 FOR J=1 TO 8
5770 LI=LEN(IV$(J,1)):REM LONGITUD OBJETO
5780 FORI=1TO LI-LW+1
5790 IFMID$(IV$(J,1),I,LW)=W$THENF=J:I=LI:J=8:K=LN
5800 NEXT I,J,K
5810 RETURN
5820 :
5830 REM **** S/R LLEVA JUGADOR EL OBJETO ****
5840 HF=0
5850 IF IV$(OV,2)="-1"THEN HF=1
5860 RETURN

2540 REM **** S/R LISTAR INVENTARIO ****
2550 PRINT"OBJETOS QUE LLEVAS CONTIGO:"
2560 FOR I=1 TO 4
2570 PRINT" ";IC$(I)
2580 NEXT I
2590 RETURN
```

Complementos al BASIC

Spectrum:

En el listado de *El bosque encantado*, introduzca las siguientes modificaciones:

Sustituya SN$ por S$, IV$(,) por V$(,), IC$() por I$() y NN$ V$(,) por R$.

```
5320 IF R$(K TO K)<>" " THEN NEXT K:RETURN
5325 LET W$=R$(C TO K-1)
5360 IF V$(I TO I+LW-1)=W$ THEN LET
     F=J:LET I=LI:LET J=3:LET K=LN
```

En el listado de *Digitaya*, reemplace los nombres de las mismas variables en serie e introduzca los mismos cambios indicados arriba, pero para las líneas 5750, 5755 y 5790, respectivamente

Jugada audaz

He aquí un nuevo e interesante microordenador "de regazo": el Osborne Encore, compatible con el IBM-PC

Al haber sido el primer ordenador portátil todo en uno, el Osborne-1 marcó el inicio de una revolución en la microinformática. Equipado con un monitor incorporado, unidades de disco gemelas e interfaces para modems e impresoras, la máquina fue el primer ordenador de oficina CP/M autocontenido. Si bien la calificación de "portátil" quizá haya sido un tanto exagerada (el Osborne-1 pesaba 10,5 kg), otras firmas informáticas comprendieron muy rápidamente el potencial de la nueva máquina, y su fabricante, la empresa norteamericana Osborne Corporation, en vez de verse en un campo propio, se encontró rodeada de competidores.

En 1981 la nueva empresa recibió un duro golpe al anunciar IBM el lanzamiento del primer modelo de su gama Personal Computer. La nueva máquina enseguida barrió a todas las otras, porque los hombres de negocios se apresuraron a adquirir el modelo del conocido gigante de la industria informática. Osborne, junto con muchos de sus competidores, anunció rápidamente el inminente lanzamiento de una máquina compatible con el IBM-PC. Este nuevo modelo, el Osborne Executive, iba a estar equipado con procesadores duales, lo que le permitiría ejecutar tanto software CP/M como MS-DOS. Sin embargo, debido a la escasez mundial de microprocesadores 8086, la máquina salió al mercado sin el chip necesario para que fuera compatible con el IBM-PC. A consecuencia de ello, las ventas de los microordenadores Osborne descendieron bruscamente, y la firma norteamericana se vio obligada a presentar liquidación voluntaria en el verano de 1983. Pero Osborne logró sobrevivir y el estilo de funcionamiento de la nueva empresa es hoy similar al de Sinclair Research en Gran Bretaña, orientada hacia la investigación y el desarrollo.

Finalmente ha aparecido una nueva máquina producida por la empresa, el Osborne Encore. Es una jugada temeraria de una firma que apenas si ha conseguido permanecer en el negocio. A pesar de uno ser tan compacta como máquinas como el Epson PX-8, la importancia del Encore radica en su intento por cubrir el vacío que existe en el mercado de gestión entre el sector de máquinas "de regazo" y el de las de escritorio.

Con un peso de 6 kg, el Encore es mucho más ligero que su predecesor. El teclado se cierra contra la pantalla, conformando una caja compacta cuyo tamaño es aproximadamente el de tres listines telefónicos. La carcasa es de plástico duro de color azul y el teclado se mantiene en su sitio mediante un par de clips.

El teclado consta de teclas QWERTY tipo máquina de escribir, encima de las cuales hay una membrana plástica que comprende las teclas de función y los "iconos" (símbolos que representan a los programas incorporados). En el cuerpo principal del ordenador propiamente dicho hay una pantalla de visualización en cristal líquido que mide 23,7 × 8 cm.

El teclado ofrece un tacto sólido y profesional, con las teclas de control a ambos lados y cuatro teclas para el cursor en el extremo inferior derecho. El diseño del teclado es muy apretado y esto, sin duda, es consecuencia de incluir en un espacio tan limitado todas las características del IBM-PC. En el lado derecho del IBM hay un teclado separado que proporciona las funciones de calculadora. Para mantener la compatibilidad en el Encore, estas teclas se han incorporado en el cuerpo principal del teclado. Las funciones de calculadora están marcadas en color azul, contrastando con el etiquetado blanco de las teclas alfanuméricas estándares. A la calculadora, así como a los otros programas incorporados del Encore, se accede pulsando el icono adecuado en el panel que hay encima del teclado. Los iconos del Encore representan las rutinas de calculadora, modem, disco y calendario.

El panel sensible al tacto no está tan bien diseñado y no posee el mismo tacto profesional que ofrece globalmente el teclado. Las teclas de función (que en el IBM-PC están situadas aparte, en el sector izquierdo) poseen el mismo tipo de tacto que el teclado del ZX81. A pesar de que uno puede percibir el "pop" de la membrana de burbuja por debajo del panel, las teclas carecen de la firmeza del tacto de un teclado adecuado.

Con la incorporación de una pantalla LCD, los

Paquete compacto
El Osborne Encore es una de las primeras máquinas compatibles con el IBM-PC que viene equipada con una visualización LCD en lugar del tubo de rayos catódicos estándar. Cuando no se lo utiliza, el teclado se repliega contra la pantalla, conformando un paquete muy compacto que se puede transportar mediante la correa suministrada para colgar del hombro

Chris Stevens

diseñadores han efectuado el mayor ahorro en cuanto a consumo de energía, dado que una visualización en cristal líquido utiliza muchísima menos electricidad que el habitual tubo de rayos catódicos. Es aquí donde se presentan los problemas más grandes de compatibilidad con el IBM-PC. No es que la pantalla LCD carezca de colores, puesto que éstos se pueden sustituir fácilmente variando la tonalidad de los gráficos: es el tamaño de la pantalla lo que constituye el principal inconveniente.

La pantalla IBM normal tiene una resolución de texto de 80 por 25 caracteres; pero, debido a problemas de desarrollo que se les plantearon a los fabricantes japoneses de la pantalla para producir la visualización LCD equivalente, Osborne se ha visto obligada a introducir el Encore con una visualización de 80 por 16. Esto significa que muchos paquetes escritos para el IBM no se podrán utilizar con el Encore. Esto no representa ningún problema para los programas que desplazan sus líneas en la pantalla, pero en los paquetes en los cuales la visualización está "paginada", el usuario puede tropezar con serias dificultades, especialmente teniendo en cuenta que los mensajes aparecen por lo general en la parte inferior de la pantalla.

El lado derecho del ordenador se ha preparado para un par de unidades de disco flexibles de 5 ¼ pulgadas, si bien el modelo estándar viene equipado con una única unidad. En el lado opuesto está el conector de potencia para el transformador, un interruptor on/off, una perilla para regular el contraste de la pantalla y la caja de las pilas, que puede albergar un paquete especial de pilas de níquel-cadmio para poder hacer funcionar la máquina cuando no se tiene acceso a un enchufe eléctrico.

En la parte trasera del Encore están las puertas de E/S. De izquierda a derecha, éstas comprenden un enchufe hembra telefónico para conectar el modem incorporado del Encore, una puerta Centronics para conectar en interface con una impresora y una puerta en serie RS232 para conectar con dispositivos como impresoras en serie y modems.

Para cargar el sistema desde el disco MS-DOS basta pulsar el icono del disco en teclado sensible al tacto. Los programas incorporados se pueden ejecutar en cualquier momento, independientemente de qué programa se esté ejecutando en ese momento en la unidad de disco.

El ordenador leyó todos los discos de programas IBM que se cargaron en el Encore. No obstante, debido a las restricciones de la pantalla, resultó difícil detectar si los programas se estaban ejecutando correctamente, dado que muchas de las instrucciones se entraron "ciegas". Entre los programas que el Encore consiguió ejecutar con total éxito estaba el *Lotus 1-2-3*: un programa difícil de ejecutar en cualquier compatible debido a la forma en que accede a las rutinas incorporadas del IBM.

Resulta difícil decir si el Encore se convertirá en una máquina con tanto éxito como el Osborne-1. Acostumbrarse a la pantalla lleva algo de tiempo ya que, al igual que todas las visualizaciones LCD, necesita una luz muy intensa para producir caracteres que sean suficientemente legibles. Éste es un problema irrelevante en máquinas de regazo con caracteres grandes, pero el tamaño de los tipos del Encore es de aproximadamente la mitad, y es difícil saber si muchos usuarios querrán invertir el tiempo necesario en acostumbrarse a la visualización.

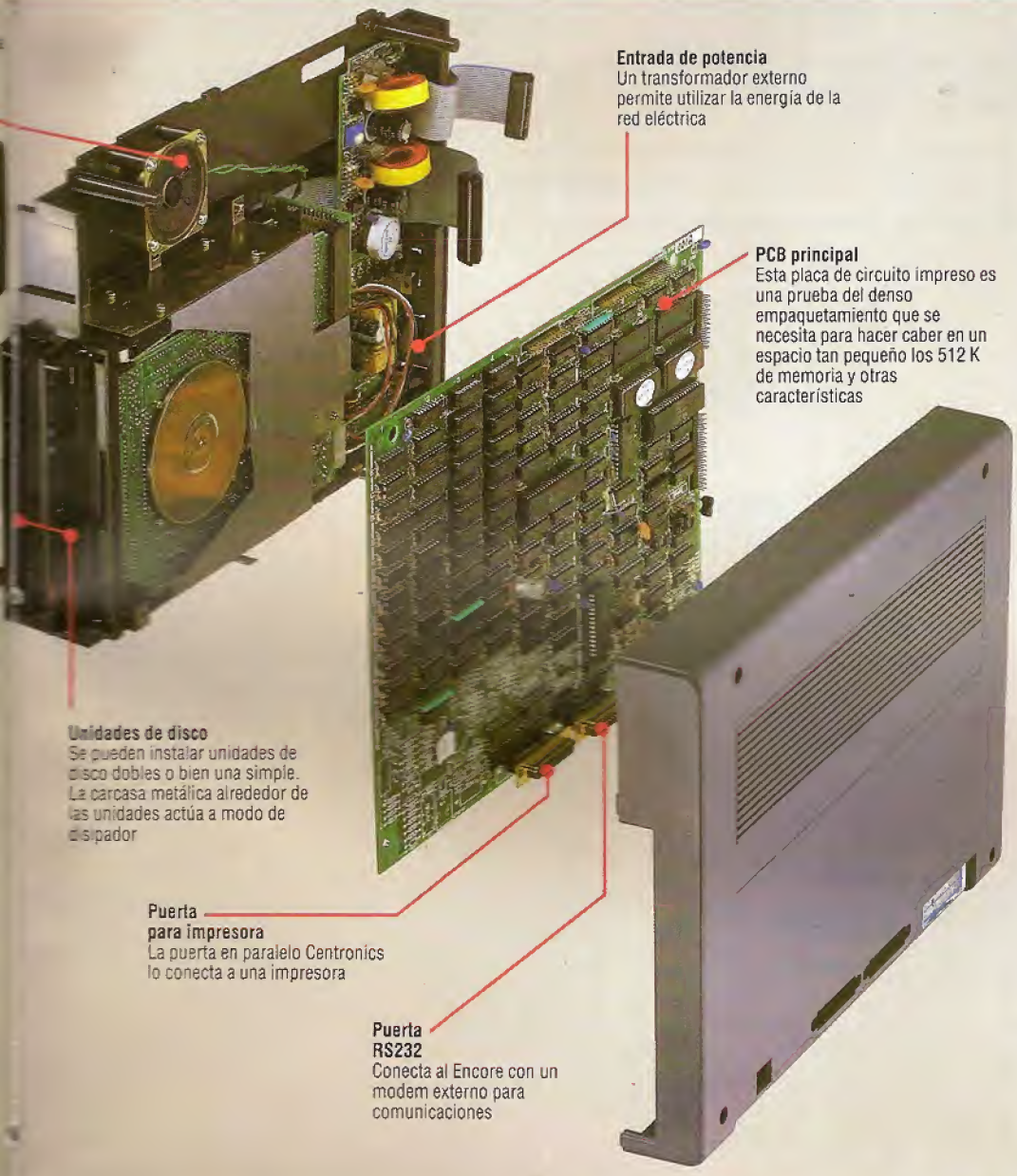
Altavoz
Las indicaciones de la pantalla se acompañan con un *beep*

Pantalla LCD
La visualización en cristal líquido permite que el Encore consuma muchísima menos potencia que una máquina provista de un tubo de rayos catódicos estándar, haciendo de esta máquina compatible con el IBM y que funciona a pilas una propuesta realista

Duplicando
El teclado del Encore posee menos teclas que el IBM-PC, pero ofrece las mismas funciones. Para conseguirlo, algunas de las teclas (señaladas con inscripciones en azul) también sirven como teclas de calculadora

Desventaja del disco

Las unidades de disco (se pueden instalar unidades de disco dobles o bien una sola) están situadas a uno de los lados de la máquina. Ello contribuye a hacer del Encore un paquete muy compacto, pero el usuario se ve forzado a estirarse para insertar un disco o comprobar si se está accediendo a la unidad adecuada



Entrada de potencia
Un transformador externo permite utilizar la energía de la red eléctrica

PCB principal
Esta placa de circuito impreso es una prueba del denso empaquetamiento que se necesita para hacer caber en un espacio tan pequeño los 512 K de memoria y otras características

Unidades de disco
Se pueden instalar unidades de disco dobles o bien una simple. La carcasa metálica alrededor de las unidades actúa a modo de disipador

Puerta para impresora
La puerta en paralelo Centronics lo conecta a una impresora

Puerta RS232
Conecta al Encore con un modem externo para comunicaciones

Chris Stevens

OSBORNE ENCORE

DIMENSIONES
241×325×141 mm

CPU
Microprocesador de 16 bits 8086

MEMORIA
Se suministra con 512 K de RAM

PANTALLA
Visualización LCD de 80×16 caracteres de 480×128 pixels. Osborne ha anunciado una pantalla de 80×25. Se les ha prometido a los usuarios una actualización económica para incorporar la pantalla más grande

INTERFACES
Interface en serie RS232, enchufe hembra telefónico RJ11 y puerta en paralelo Centronics

LENGUAJES
BASIC Microsoft en disco

TECLADO
62 teclas tipo máquina de escribir y un teclado que contiene 10 teclas de función y cuatro teclas de iconos

DOCUMENTACION
Las dos guías para el usuario están, como todos los manuales de Osborne, notablemente bien escritas; ofrece una completa explicación acerca de cómo utilizar la máquina y contiene muchísimos ejemplos

VENTAJAS
Es una máquina sumamente potente y compacta que puede operar a pilas entre cuatro y cinco horas

DESVENTAJAS
La pantalla todavía no es totalmente compatible con el IBM-PC y puede resultar difícil de leer

Control de tiempo
En la fotografía vemos la pantalla LCD de 80 × 16 caracteres. Las ranuras de debajo de la pantalla harán posible instalar una visualización de 80 × 25. También se puede observar el calendario/diario, que permite al operador entrar las futuras citas y establecer la hora en cualquiera de las zonas del mapamundi

Sensible al tacto
Encima del teclado principal hay un teclado sensible al tacto (similar al teclado del Spectrum y del ZX81) que contiene las 10 teclas de función programables. Debajo de éstas están los iconos que se utilizan para llamar a los programas basados en ROM

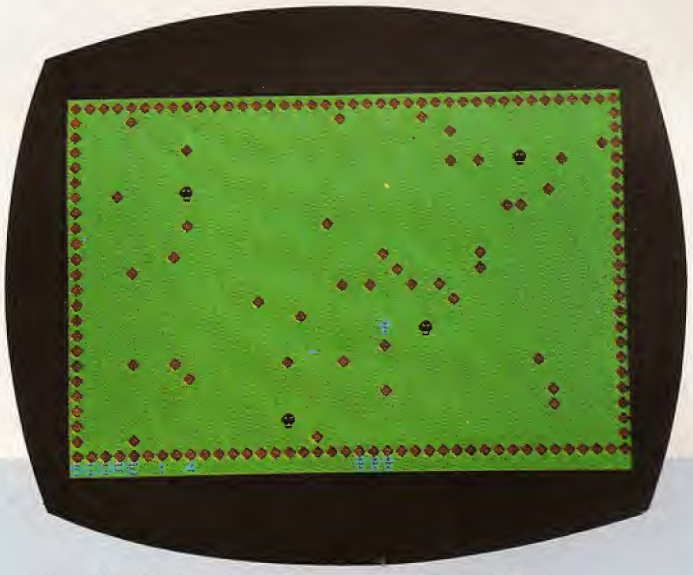
Robots

Usted se encuentra solo, abandonado en un desconocido planeta defendido por robots asesinos. He aquí un emocionante programa para el micro Thomson TO 7

Las minas están representadas en la pantalla mediante rombos rojos. Al comenzar el juego, cinco robots se hallan sobre el terreno. Sin perder un segundo, se precipitan hacia usted, siguiendo siempre el camino más corto. Por suerte los robots son ciegos y no pueden ver las minas situadas entre usted y ellos, lo cual le permitirá, siempre que se desplace adecuadamente, eliminarlos. Para ello utilice la palanca de mando o las teclas

```
< A >   < Z >   < E >
< Q >           < D >
< W >   < X >   < C >
```

según la dirección elegida por usted. Cuando haya eliminado a todos los robots, el juego continúa con un robot suplementario. Si salta sobre una mina o un robot le mata, aún no está todo perdido. En realidad tiene cinco vidas. Si desea cambiar el número de minas, modifique el valor de la variable NM en la línea 80.



```
10 REM ********************************
20 REM *            ROBOTS            *
30 REM ********************************
40 DEFINT A-Z
50 CLEAR ,,3
60 NH=5
70 N1=5
80 NM=40
90 NR=N1
100 DIM R(30,1)
110 GOSUB 1580
120 GOSUB 1470
130 GOSUB 910
140 ON JS GOSUB 710,810,
150 C=POINT (HX*8+4,HY*8+4)
160 IF C<>-3 AND C<>4 THEN 470
170 COLOR 4
180 LOCATE X,Y
190 PRINT N$;
200 LOCATE HX,HY
210 PRINT H$;
220 X=HX
230 Y=HY
240 T=0
250 FOR I=1 TO NR
260 IF R(I,0)=0 THEN 400
270 T=1
280 RX=R(I,0)+SGN(HX-R(I,0))
290 RY=R(I,1)+SGN(HY-R(I,1))
300 C=POINT (RX*8+4,RY*8+4)
310 IF C=1 OR C=0 THEN S=S+1:LOCATE
    R(I,0),R(I,1):PRINT N$;:R(I,0)=0:GOTO 400
320 IF C=4 THEN 470
330 COLOR 0
340 LOCATE (R(I,0),R(I,1)
350 PRINT N$;
360 LOCATE RX,RY
370 PRINT R$;
380 R(I,0)=RX
390 R(I,1)=RY
400 NEXT I
410 IF T=0 THEN 430
420 GOTO 140
430 S=S+10
440 IF INKEY$<>"" THEN 440
450 IF NR<30 THEN NR=NR+1
460 GOTO 130
470 NH=NH-1
480 COLOR 7
490 LOCATE X,Y
500 PRINT N$;
510 LOCATE HX,HY
520 PRINT H$;
530 PLAY "L96REL72REL24REL96REL72FAL24MI
    L72MIL24REL72REL24DO#L96RE"
540 IF INKEY$<>"" THEN 540
550 IF NH>0 THEN NR=N1:GOTO 130
560 CLS
570 SCREEN 1,6,6
580 ATTRB 1,1
590 LOCATE 9,10
600 PRINT "PUNTUACION :";S;
610 LOCATE 9,20
620 PRINT "OTRA ?";
630 COLOR 4
640 ATTRB 0,0
650 IF INKEY$<>"" THEN 650
660 D$=INKEY$
670 IF D$="" THEN 660
680 IF D$<>"N" THEN RUN
690 CLS
700 END
710 D$=INKEY$
720 IF D$="A" THEN HX=HX-1:HY=HY-1
730 IF D$="Z" THEN HY=HY-1
740 IF D$="E" THEN HY=HY-1:HX=HX+1
750 IF D$="Q" THEN HX=HX-1
760 IF D$="D" THEN HX=HX+1
770 IF D$="W" THEN HX=HX-1:HY=HY+1
780 IF D$="X" THEN HY=HY+1
790 IF D$="C" THEN HY=HY+1:HX=HX+1
800 RETURN
810 J=STICK(0)
820 IF J=1 THEN HY=HY-1
830 IF J=2 THEN HY=HY-1:HX=HX+1
840 IF J=3 THEN HX=HX+1
850 IF J=4 THEN HX=HX+1:HY=HY+1
860 IF J=5 THEN HY=HY+1
870 IF J=6 THEN HY=HY+1:HX=HX-1
880 IF J=7 THEN HX=HX-1
890 IF J=8 THEN HX=HX-1:HY=HY-1
900 RETURN
910 CLS
920 COLOR 4
930 LOCATE 0,24
940 PRINT "PUNTUACION :";S;
950 IF NH=1 THEN 1000
960 FOR HX=1 TO NH-1
970 LOCATE 19+HX,24
980 PRINT H$;
990 NEXT HX
1000 COLOR 1
1010 FOR HX=0 TO 39
1020 LOCATE HX,0
1030 PRINT M$;
1040 LOCATE HX,23
1050 PRINT M$;
1060 NEXT HX
1070 FOR HY=1 TO 22
1080 LOCATE 0,HY
1090 PRINT M$;
1100 LOCATE 39,HY
1110 PRINT M$;
1120 NEXT HY
1130 FOR I=1 TO NM
1140 HX=INT(RND*38)+1
1150 HY=INT (RND*22)+1
1160 IF SCREEN(HX,HY)<>32 THEN 1140
1170 LOCATE HX,HY
1180 PRINT M$;
1190 NEXT I
1200 COLOR 0
1210 FOR I=1 TO NR
1220 R(I,0)=INT (RND*38)+1
1230 R(I,1)=INT (RND*22)+1
1240 IF SCREEN(R(I,0),R(I,1))<>32 THEN 1220
1250 LOCATE R(I,0),R(I,1)
1260 PRINT R$;
1270 NEXT I
1280 HX=INT (RND*38)+1
1290 HY=INT (RND*22)+1
1300 IF SCREEN(HX,HY)<>32 THEN 1280
1310 X=HX
1320 Y=HY
1330 FOR I=1 TO 5
1340 LOCATE HX,HY
1350 COLOR 5
1360 PRINT CHR$(127);
1370 BEEP
1380 FOR J=1 TO 50
1390 NEXT J
1400 LOCATE HX,HY
1410 COLOR 4
1420 PRINT H$;
1430 FOR J=1 TO 50
1440 NEXT J
1450 NEXT I
1460 RETURN
1470 CLS
1480 SCREEN 4,2,0
1490 ATTRB 1,1
1500 LOCATE 10,10,0
1510 PRINT "JOYSTICK ?";
1520 ATTRB 0,0
1530 D$=INKEY$
1540 C=RND
1550 IF D$="" THEN 1530
1560 IF D$="S" THEN JS=2 ELSE JS=1
1570 RETURN
1580 DEFGR$(0)=28,28,73,62,8,28,20,20
1590 DEFGR$(1)=60,126,219,255,255,126,36,60
1600 DEFGR$(2)=0,0,24,60,126,126,60,24
1610 H$=GR$(0)
1620 R$=GR$(1)
1630 M$=GR$(2)
1640 N$=CHR$(32)
1650 RETURN
```

Sesión continua

En este capítulo crearemos nuevas estructuras de control y haremos uso de las capacidades recursivas de este lenguaje

La primitiva RUN del LOGO toma una lista como entrada y hace que se ejecute como si fuera una línea de un procedimiento. Se la puede utilizar para agregarle al lenguaje nuevas estructuras de control cómo y cuándo se las requiera. De manera que podríamos definir un procedimiento MIENTRAS del siguiente modo:

```
TO MIENTRAS :CONDICION :ACCION
  IF NOT (RUN :CONDICION) THEN STOP
  RUN :ACCION
  MIENTRAS :CONDICION :ACCION
END
```

Veamos ahora un ejemplo de cómo podríamos utilizarlo. POTENCIA imprime todas las potencias de su entrada inferiores a 1000:

```
TO POTENCIA :X
  MAKE "P :X
  MIENTRAS [:P<1000][PRINT :P MAKE "P :P*:X]
END
```

Las estructuras de control, como WHILE, REPEAT y FOR, son comunes en otros lenguajes, pero en LOGO no son realmente necesarias. Una forma más natural de escribir POTENCIA sería:

```
TO POTENCIA :P
  IF NOT :P<1000 THEN STOP
  PRINT :P
  POTENCIA P*:P
END
```

No todas las versiones de LOGO disponen de REPEAT, pero esta estructura no es realmente necesaria, ya que se podría definir una palabra equivalente, REP, del siguiente modo:

```
TO REP :NUM :LISTA
  IF :NUM=0 THEN STOP
  RUN :LISTA
  REP :NUM-1 :LISTA
END
```

RUN es una primitiva sumamente útil para trabajos más avanzados. Un programa puede ensamblar una lista y pasársela luego a RUN para que se la ejecute. Enseguida veremos un ejemplo de esto.

Desarmar procedimientos

En primer lugar debemos definir un procedimiento para dibujar un triángulo de la forma habitual:

```
TO TRI
  FD 50 RT 120 FD 50
  RT 120 FD 50 RT 120
END
```

Ahora digite PRINT TEXT "TRI. El resultado será:

```
[] [FD 50] [RT 120] [FD 50] [RT 120] [FD 50] [RT 120]
```

El texto del procedimiento se da como una lista de listas, donde cada lista "interior" es una línea del procedimiento. Para ver por qué al principio hay una lista vacía, defina esta variante para la suma:

```
TO SUMAR :A :B
  PRINT :A+:B
END
```

Ahora PRINT TEXT "SUMAR dará:

```
[:A :B][PRINT :A+:B]
```

Evidentemente, la primera lista contiene las entradas para el procedimiento. De modo que TEXT nos permite introducirnos en un procedimiento y averiguar qué hay allí. DEFINE, por el contrario, hace exactamente lo inverso: nos permite definir un procedimiento como una lista de listas sin tener que acudir al editor. Pruebe ahora DEFINE "L[[:A][FD :A][RT 90][FD :A/2]] y después ejecute L utilizando, por ejemplo, L 30. El empleo de DEFINE de esta forma en modo inmediato no ofrece ninguna ventaja respecto al empleo del editor. La ventaja que nos reporta DEFINE es la posibilidad de que un procedimiento cree otro procedimiento.

Crecimiento

Ahora vamos a desarrollar un pequeño sistema para investigar el crecimiento. Las instrucciones básicas de nuestro sistema son PEDIR, que selecciona la forma de la que nos ocuparemos, y CRECER, que cambia el tamaño de la forma elegida. Por ejemplo, PEDIR "CUADRADO dibujará un cuadrado, y luego CRECER[* 10] borrará el cuadrado y volverá a dibujarlo con cada uno de sus lados incrementado en un factor de 10.

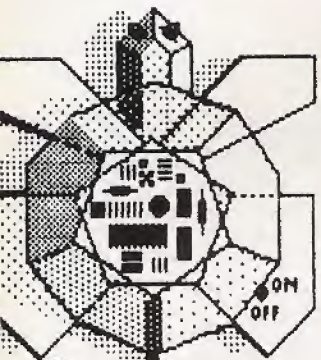
Para que los programas sean sencillos tendremos que aceptar algunas restricciones en relación al uso de estas instrucciones. En primer lugar, los procedimientos de formas dados a PEDIR como entrada no han de contener REPEAT ni llamar a subprocedimientos. En segundo lugar, el sistema se colgará si se obtienen resultados negativos. Resolver estos problemas no es muy difícil si usted desea introducir mejoras en lo que le ofreceremos aquí.

PEDIR funciona asignándole el nombre de la forma a la variable global "ACTUAL y ejecutando luego el procedimiento. Esto lo hace creando una lista de un elemento (el nombre del procedimiento) y valiéndose después de RUN para ejecutarla.

```
TO PEDIR :OBJETO
  HIDETURTLE
  MAKE "ACTUAL :OBJETO
  RUN (LIST :OBJETO)
END
```

CRECER borra primero el dibujo original (para el borrado el LOGO Commodore usa PENCOLOR-1), luego utiliza DEFINE para definir al procedimiento

Dibújame una tortuga
No se puede avanzar mucho en Logo sin encontrarse con la recursión, algo que se define en función de sí mismo. Hemos visto ejemplos como procedimientos que se llaman a sí mismos, listas definidas en términos de listas y ahora procedimientos que escriben procedimientos. Con un poco de imaginación, en Logo sería fácil crear un dibujo que utilizara a la tortuga para generar una tortuga que dibujara una tortuga...



actual como reescrito. El color del lápiz vuelve luego a ser normal y se dibuja la nueva forma. Observe que la entrada de CRECER se almacena en OPLIST, que nos será necesaria después.

```
TO CRECER :OPLIST
  PENCOLOR-1
  RUN (LIST :ACTUAL)
  DEFINE :ACTUAL REESCRIBIR.PROC TEXT
    :ACTUAL
    PENCOLOR 1
  RUN (LIST :ACTUAL)
END
```

REESCRIBIR.PROC separa el texto en líneas y se las va pasando de una en una a REESCRIBIR.LINEA:

```
TO REESCRIBIR.PROC :TEXTO
  IF EMPTY? :TEXTO THEN OUTPUT []
  OUTPUT FPUT REESCRIBIR.LINEA FIRST
    :TEXTO REESCRIBIR.PROC BUTFIRST :TEXTO
END
```

REESCRIBIR.LINEA busca a lo largo de una línea una FD o FORWARD. De encontrar una, le pasa el resto de la línea a CAMBIAR para que se encargue de ella.

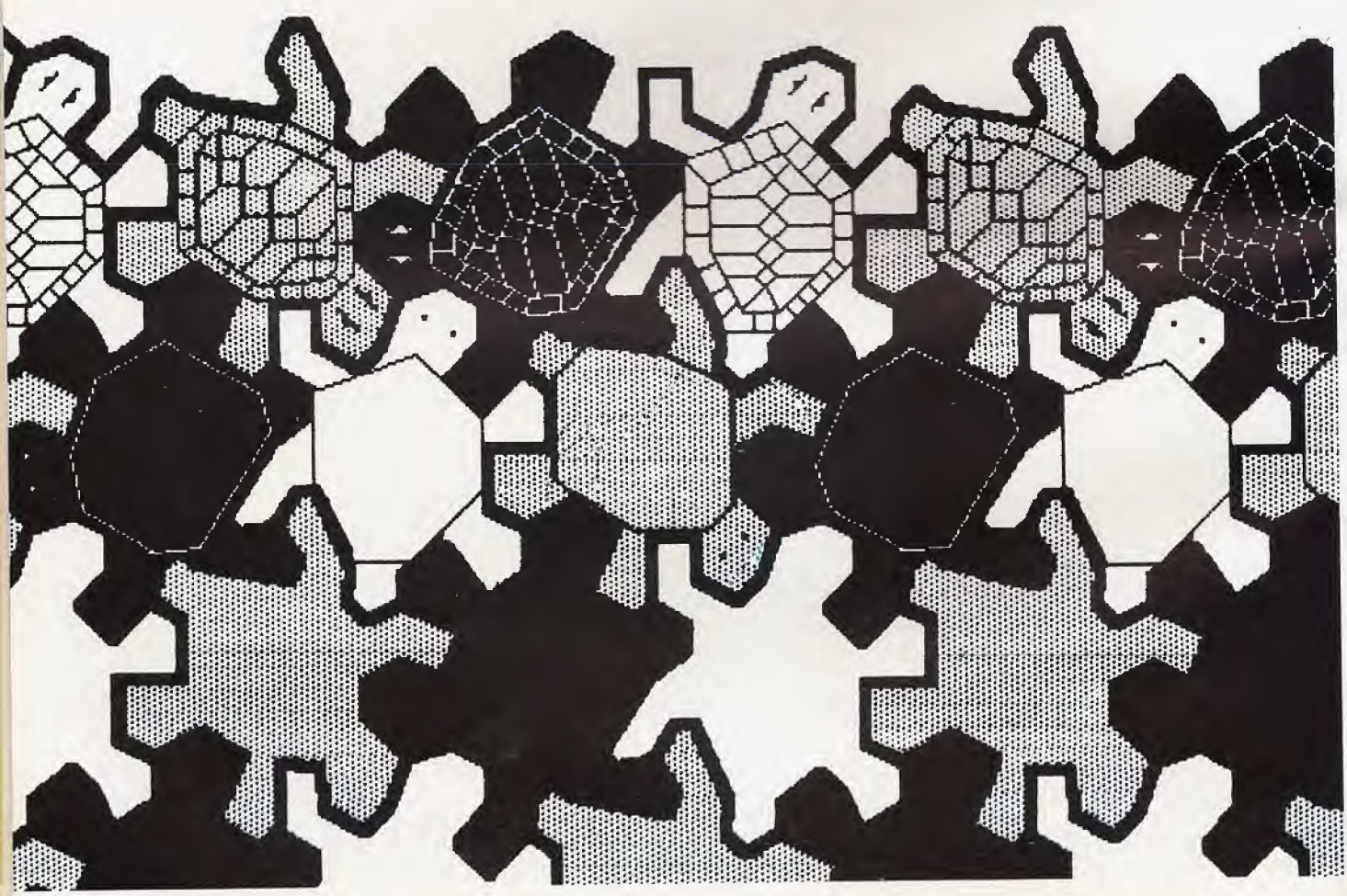
```
TO REESCRIBIR.LINEA :LINEA
  IF EMPTY? :LINEA THEN OUTPUT []
  IF ANYOF FIRST :LINEA="FD FIRST :LINEA=
    "FORWARD THEN OUTPUT CAMBIAR BUT-
FIRST
    :LINEA
  OUTPUT FPUT FIRST :LINEA REESCRIBIR.
    LINEA BUTFIRST :LINEA
END
```

CAMBIAR construye la línea "reescrita". El primer elemento de LISTA (la entrada de CAMBIAR) habrá sido la entrada de FORWARD en el procedimiento original. Supongamos que es 50 y, si OPLIST contuviera [* 2], entonces SENTENCE FIRST :LISTA :OPLIST sería [50 * 2]. Ahora CAMBIAR utiliza RUN para evaluar esta lista (obteniendo un resultado de 100). Por último, se construye una lista compuesta por FD, la cantidad que se acaba de evaluar, y después la reescritura del resto de la línea:

```
TO CAMBIAR :LISTA
  OUTPUT(SENTENCE "FD(RUN SENTENCE
    FIRST :LISTA :OPLIST) REESCRIBIR.
    LINEA BUTFIRST :LISTA)
END
```

Imitador

En algunas ocasiones es útil poder hacer una copia de un procedimiento. De esta manera, vamos a definir un procedimiento (COPIARDEF), de modo que COPIARDEF "NUEVONOMBRE "VIEJONOMBRE defina NUEVONOMBRE como una copia de VIEJONOMBRE (VIEJONOMBRE no sufrirá ninguna alteración). Una definición obvia es:



```
TO COPIARDEF :NUEVO :VIEJO
  DEFINE :NUEVO TEXT :VIEJO
END
```

El problema de esta definición es que si VIEJO no existe, el procedimiento se limitará a seguir adelante y definirá a NUEVO como nada. Sería mejor detectar este problema e informar sobre el mismo. Por lo tanto, una definición mejor de COPIARDEF sería:

```
TO COPIARDEF:NUEVO :VIEJO
  IF NOT PROCEDIMIENTO? :VIEJO THEN
  (PRINT[NO HAY NINGUN PROCEDIMIENTO]
  :VIEJO)STOP
  DEFINE :NUEVO TEXT :VIEJO
END
```

Aquí se utiliza un procedimiento llamado PROCEDIMIENTO?, que produce VERDADERO si su entrada es un procedimiento, y, de lo contrario, FALSO. PROCEDIMIENTO? y su complemento, PRIMITIVA?, son comprobaciones muy útiles; pero lamentablemente no existen en LOGO MIT. De modo que hemos desarrollado versiones de PROCEDIMIENTO? y PRIMITIVA? que funcionarán con las versiones LOGO tanto Apple como Commodore:

```
TO PROCEDIMIENTO? :NOMBRE
  IF NUMBER? :NOMBRE THEN OUTPUT
    "FALSE IF LIST? :NOMBRE THEN OUTPUT
      "FALSE
  TEST WORD? :NOMBRE
  IFTRUE IF WORD? TEXT :NOMBRE THEN
    OUTPUT "FALSE ELSE IF NOT (TEXT
    :NOMBRE=[]) THEN OUTPUT "TRUE
    OUTPUT "FALSE
END

TO PRIMITIVA? :NOMBRE
  IF NUMBER? :NOMBRE THEN OUTPUT
    "FALSE
  IF LIST? :NOMBRE THEN OUTPUT "FALSE
  TEST WORD? :NOMBRE
  IFTRUE IF WORD? TEXT :NOMBRE THEN
    OUTPUT "TRUE ELSE OUTPUT "FALSE
END
```

Últimas palabras

Ahora ya hemos visto las características más importantes del LOGO estándar y hemos cubierto una amplia área de posibles aplicaciones. Si desea leer algo más sobre el lenguaje, a continuación le sugerimos cuatro libros:

- *Learning with LOGO*, de Daniel Watt (McGraw-Hill), es un maravilloso libro de introducción e ideal para utilizarlo con niños.
- *LOGO*, de Harold Abelson (McGraw-Hill), es el libro "estándar" sobre el lenguaje.
- *Turtle geometry*, de Harold Abelson y Andrea diSessa (MIT Press) realiza un serio análisis sobre la geometría de tortuga. La matemática involucrada es de un nivel de primer curso y de facultad; ¡en uno de los últimos capítulos se desarrolla un simulador para relatividad general en LOGO!
- *Thinking about [TLC] LOGO*, de John R. Allen, Ruth E. Davis y John F. Johnson (Holt Sanders International Editions), utiliza el TLC LOGO, que es bastante idiosincrásico, pero la obra es valiosa por sus temas de investigación y de inteligencia artificial en LOGO.

Las virtudes del LOGO

- Es interpretado, al igual que el BASIC, de modo que es fácil ejecutar y modificar programas.
- Es estructurado, con auténticos procedimientos, a diferencia de la mayor parte de las versiones de BASIC.
- Es ampliable, como el FORTH; es decir, es posible definir palabras nuevas que luego pasen a formar parte del vocabulario del ordenador.
- Posee procesamiento de listas, al igual que el LISP, por lo cual es útil para la exploración de áreas tales como la inteligencia artificial.

El LOGO en realidad no está diseñado para implementar algoritmos perfectos totalmente elaborados, pero es ideal para trabajos no especialmente "delicados". Hemos escrito la mayoría de los programas de esta serie partiendo de un procedimiento sencillo para llevar a cabo apenas una parte de la tarea. Luego alteramos el procedimiento de diversas maneras y, a consecuencia de ello, lo mejoramos y lo desarrollamos. Al final nos encontrábamos con un algoritmo perfecto bien diseñado.

Carencias del LOGO

- Las principales son el limitado espacio de trabajo y la lenta velocidad de ejecución.
- Serían útiles otras características; en particular, carece de facilidades para depuración de errores y de manipulación de matrices y archivos.

Complementos al LOGO

En todas las versiones LCSI realice las siguientes sustituciones:

NUMBERP por NUMBER?
LISTP por LIST?
WORDP por WORD?
EMPTYP por EMPTY?

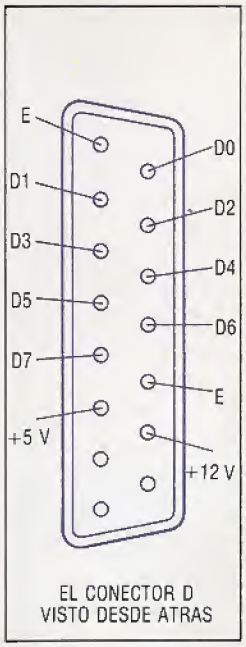
El LOGO Spectrum dispone de COPYDEF (COPIARDEF) como primitiva, así como PRIMITIVEP (por PRIMITIVA?) y DEFINEDP (por PROCEDIMIENTO?).

En el Atari utilice: PC por PENCOLOR, SE por SENTENCE y HT por HIDETURTLE. DEFINE y TEXT no existen en el LOGO Atari, si bien se ofrece una forma de definirlas.

Los colores de lápiz utilizados cambiarán de una máquina a otra. Aquí se emplea PC-1 para borrar líneas, pero algunas versiones de LOGO poseen esta facilidad en la primitiva PE (de *Pen Erase*: borrar lápiz).

El interior

En este capítulo de nuestra serie continuaremos ensamblando el robot y escribiremos un programa que compruebe la labor que hemos realizado hasta ahora



EL CONECTOR D VISTO DESDE ATRAS

Conexión de enchufes

Los diagramas del enchufe de la puerta para el usuario muestran las conexiones para cada tipo de enchufe. Los usuarios del BBC Micro deben usar un cable plano de 20 vías y un conector IDC de 20 vías de ajuste a presión. Indicando las conexiones en el extremo libre del cable, se deben pelar los 11 cables necesarios y emparejar con las conexiones de la placa de interface.

Los usuarios del Commodore 64 deben usar un conector marginal de 24 vías y un trozo corto de cable plano de 12 vías. Marque las conexiones en el enchufe y emparéjelas con las conexiones de la placa. Ya que este conector se puede insertar por cualquiera de los lados, es importante señalar de alguna forma la parte superior del enchufe.

Ahora ya podemos enchufar el cable de la interface en el robot y en la puerta para el usuario. Asimismo, se debe enchufar una fuente eléctrica CD de 12 V en el conector de 2,1 mm de la placa de interface

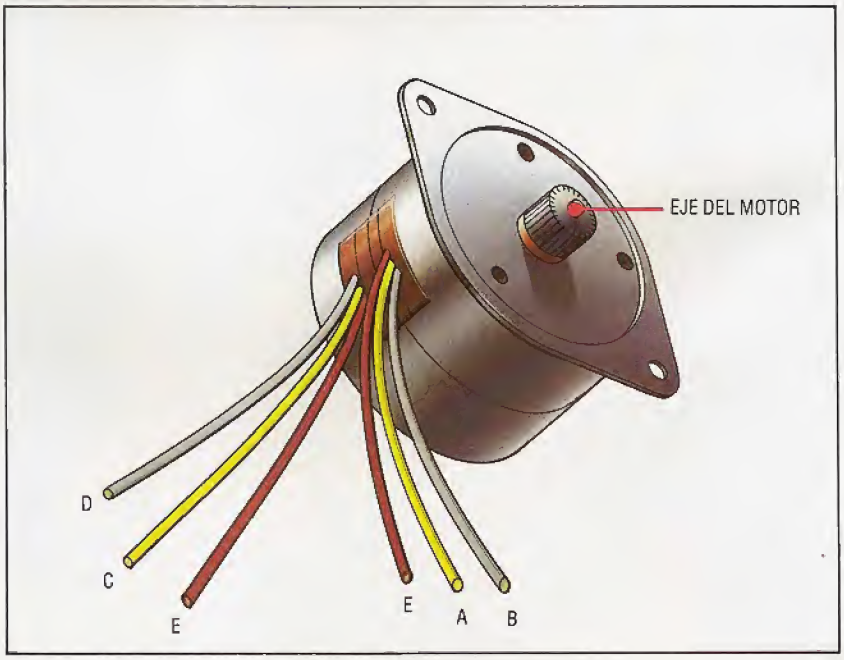
Control del robot

Ahora que ya hemos completado la primera fase de construcción, podemos escribir un corto programa para controlar el robot desde el teclado. Los bits del 0 al 3 del registro de datos de la puerta para el usuario controlan los motores. El bit 0 es el bit de inicialización, establecido normalmente en 1; los bits 1 y 2 controlan, respectivamente, las direcciones del motor a derecha e izquierda. El bit 3 es el bit de impulso que dispara los motores para que giren otro paso. El programa utiliza las teclas T, B, F y H para controlar la dirección y un bucle repetitivo para impulsar los motores

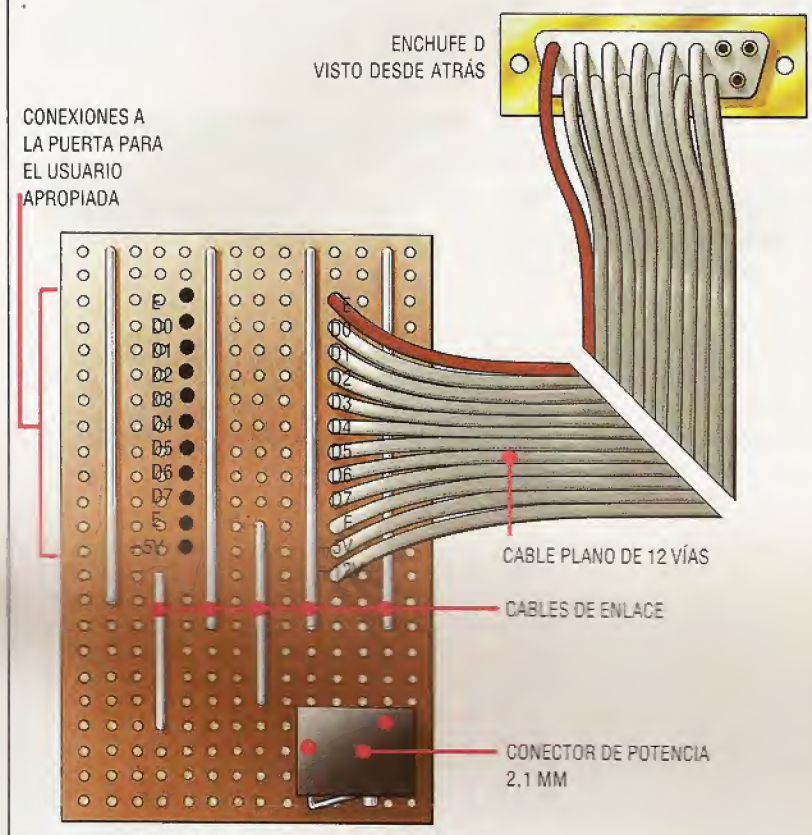
```
1000 REM **** CONTROLADOR DEL ROBOT BBC ****
1010 RDD=&FE62:REGDAT=&FE60:?RDD=15:REM LINEAS 0-3 SALIDA
1020 PROCinicializar:REPEAT
1030 A$=INKEY$(1):IF A$<>"" THEN PROCcomprobar_teclado
1040 PROCimpulso(10)
1050 UNTIL A$="X":?REGDAT=0:END
1060 DEF PROCinicializar
1070 adelante=4:atras=2:izquierda=6:derecha=0
1080 dir=adelante:?REGDAT=dir+1:ENDPROC
1100 DEF PROCimpulso(m)
1110 FOR c=1 TO m
1120 ?REGDAT=(?REGDAT OR 8):PROCdemora(2)
1130 ?REGDAT=(?REGDAT AND 247):PROCdemora(2)
1140 NEXT c:ENDPROC
1150 DEF PROCdemora(n)
1160 FOR I=1 TO n:NEXT I
1170 ENDPROC
1180 DEF PROCcomprobar_teclado
1190 IF A$="T" THEN dir=adelante
1200 IF A$="B" THEN dir=atras
1210 IF A$="F" THEN dir=izquierda
1220 IF A$="H" THEN dir=derecha
1230 ?REGDAT=((?REGDAT AND 249)OR dir)
1240 ENDPROC

10   REM **** CONTROLADOR DEL ROBOT CMB 64 ****
20   RDD=56579:REGDAT=56577:POKERDD,15
30   GOSUB1000:REM INICIALIZAR
40   GETA$:IFA$<>"" THEN GOSUB3000:REM TECLAS
50   M=10:GOSUB1500:REM IMPULSO
60   IF A$<>"X" THEN 40
70   POKEREGDAT,0:END
1000 REM **** S/R INICIALIZAR ****
1010 AD=4:AT=2:IZ=6:DE=0
1020 DR=AD:POKEREGDAT,DR+1:RETURN
1500 REM **** S/R IMPULSO ****
1510 FOR C=1 TO M
1520 POKEREGDAT,(PEEK(REGDAT)OR8):GOSUB2000:REM DEMORA
1530 POKEREGDAT,(PEEK(REGDAT)AND247):GOSUB2000:REM DEMORA
1540 NEXT C:RETURN
2000 REM **** S/R DEMORA ****
2010 FOR I=1 TO N:NEXT I:RETURN
3000 REM **** S/R PRUEBA TECLADO ****
3010 IF A$="T" THEN DR=AD
3020 IF A$="B" THEN DR=AT
3030 IF A$="F" THEN DR=IZ
3040 IF A$="H" THEN DR=DE
3050 POKEREGDAT,((PEEK(REGDAT)AND249)ORDR)
3060 RETURN
```

Eslabones perdidos

Para conectar los motores y el enchufe D a la placa de circuitos, tendrá que volver a remitirse a la ilustración de la placa de circuitos de p. 1317. En este diagrama vemos dónde soldar los cables correspondientes en la placa. Tome primero las conexiones del motor: cada motor tiene seis cables que salen de su cuerpo. Observe con atención que los cables emergen de la carcasa del motor en dos grupos de tres. Cada grupo de tres cables posee uno amarillo, uno gris y uno rojo. El par amarillo y gris etiquetado "A" y "B" sale de la carcasa desde el punto más cercano al eje del motor, tal como se aprecia en el diagrama. Relacionando estas letras con las conexiones con letras para cada motor en el diagrama de la placa de circuitos, suelde cada cable en su lugar en la placa



Kevin Jones



PLACA INTERFACE PUERTA PARA EL USUARIO

Lista de componentes

Cantidad	Artículo
1	Conector D 15 vías
1	Funda D 15 vías
1	Enchufe de potencia 2,1 mm
1	Conector IDC 20 vías (BBC)
1	Conector marginal 24 vías (C64)
1	Rollo parches autoadhesivos
Varios	
4 m	Cable plano 12 vías
1 m	Cable plano 20 vías (BBC)
1	Fuente alimentación CD 12 V 1 amp

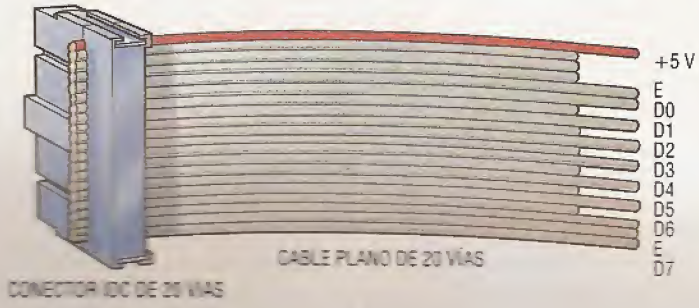
Diseñar la placa

Habiendo completado las conexiones internas para el robot, hemos de diseñar una placa interface simple que nos permita controlar al robot desde la puerta para el usuario y suministrar los 12 V de CD que requieren los motores paso a paso.

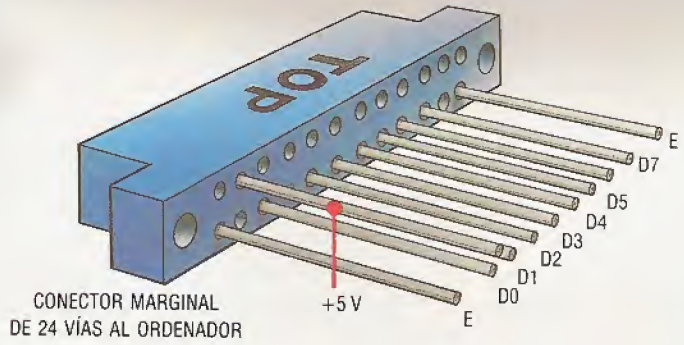
Corte un trozo de veroboard de 24 franjas por 14 agujeros y conecte 3 m de cable plano de 12 vías a la placa, tal como se indica. Utilizando la tira roja de uno de los lados del cable a modo de guía, suelde los 12 cables a las patillas correspondientes de un conector D y coloque la funda del conector D para asegurar el cable.

Monte el enchufe de potencia de 2,1 mm en la placa, señalando la orientación de las patillas. El polo central de este enchufe es negativo. Luego haga los enlaces de cables tal como se indica. Observe, también, las conexiones de la puerta para el usuario; recuerde que los conectores de ésta en el BBC Micro y el Commodore 64 son diferentes

BBC Micro



Commodore 64



Hacer conexiones

Una vez hechas las conexiones del motor, sólo nos resta hacer las conexiones adecuadas para el enchufe D, montado en la tapa del cuerpo del robot. El diagrama ilustra las conexiones de patillas correspondientes para el enchufe D, vistas desde la cara inferior de la tapa. Utilizando un trozo de cable plano de 12 vías, conecte las líneas de datos, de D0 a D3, las conexiones de +5 V, +12 V y tierra a la placa de circuitos. Deberá remitirse al diagrama de la placa de circuitos que ofrecimos anteriormente para observar las posiciones correctas del cableado de estas líneas a la placa de circuitos. Tenga especial cuidado en asegurarse de que la línea de potencia de +12 V se conecte en el punto correcto de la placa de circuitos. De no hacerlo así, se podrían dañar los circuitos internos de su ordenador. Las líneas de datos de D4 a D7 *no* se deben conectar en esta etapa, porque están reservadas para los sensores de entrada.

Todas las conexiones del interior están ahora completas. Busque dentro de la carcasa un lugar para alojar la placa de circuitos y asegúrelo con parches autoadhesivos. Cierre la tapa y fíjela con las cuatro tuercas angulares que se proporcionan

Principios operativos

Un sistema operativo hace el papel de mediador entre el programa y la máquina. Con este capítulo iniciamos el estudio de los sistemas operativos de los microordenadores más populares: BBC Micro, Sinclair Spectrum, Commodore 64 y otros

Un sistema operativo (OS: *operating system*) está escrito en el lenguaje máquina que emplea el microprocesador incorporado al ordenador. Así, por ejemplo, el OS del BBC está escrito en código máquina del 6502 y el del Spectrum en código del Z80.

Un OS se compone de una serie de rutinas a las que se confían numerosas funciones de la máquina. Según esto, puede, por ejemplo, contener una rutina que rastrea el teclado cuando se oprime una tecla, lo que permite al usuario no ocuparse de este detalle al escribir software. En un buen sistema operativo, el usuario deberá tener acceso a cualquier detalle de la máquina sin que por ello deba conocer antes su posición exacta en la memoria del ordenador o mapa de entrada/salida de la rutina que controla esa determinada parte del hardware. Esto significa que se pueden hacer más fácilmente cambios en la máquina, cuando el OS ha sido cambiado para adaptarse al nuevo hardware, de tal modo que los viejos programas puedan servir también en la nueva versión de la máquina.

El sistema operativo del BBC, por poner un ejemplo, es un modelo de planificación. Un programa escrito en BASIC del BBC para una máquina estándar servirá perfectamente en un BBC Micro equipado con un segundo procesador, aun cuando éste represente una sustancial variación del hardware de la máquina. El OS del BBC ha atravesado varias etapas de desarrollo y experimentado diversas mejoras hasta llegar a su actual estado de refinamiento. Hace tiempo que ha desaparecido del mercado la versión 0.1, que fue la primera en aparecer. Se componía de cuatro chips EPROM, pero le faltaba flexibilidad y, lo que es más grave, no preveía las unidades de disco. La que sí lo hacía era la versión 1.0, que se componía de dos chips de 8 K colocados sobre una pequeña placa de circuito impreso dentro de la máquina. La versión actual es la 1.2, que se encuentra en la mayoría de los micros BBC utilizados hoy día. Existe una variante de esta versión, la 1.2 (US), diseñada especialmente para el mercado estadounidense.

La versión de su propia máquina la puede usted averiguar fácilmente con sólo escribir la orden *FX0 y pulsar la tecla Return. El número de la versión del OS aparecerá en pantalla. La orden *HELP también proporciona el número de la versión del OS, pero lista además los nombres de los chips ROM que posee la máquina.

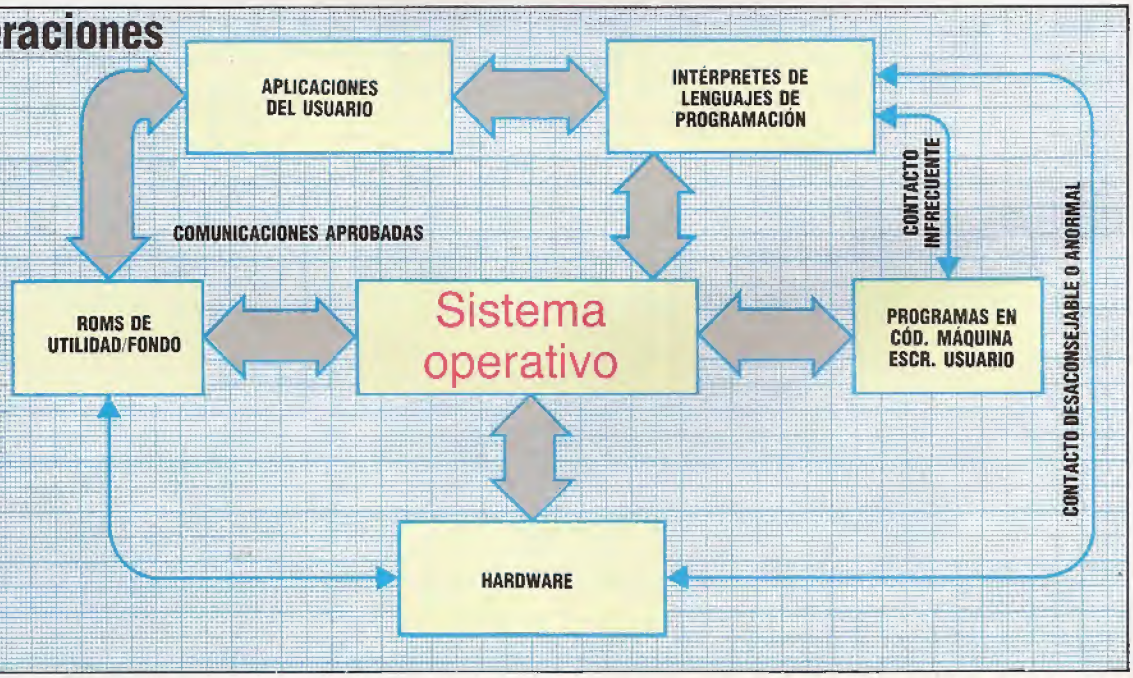
Qué hace el sistema operativo del BBC

Las tareas del sistema operativo del BBC pueden agruparse en cuatro categorías principales:

1. Rutinas de entrada: Estas rutinas reciben información de la llamada *corriente de entrada en curso*.

Teatro de operaciones

El mejor sistema operativo es el que no se ve. Para el usuario deberá ser una interface completamente cristalina entre la aplicación y el hardware, que controla discos, dosifica la salida, guía la pantalla, hace el trabajo sistemático. Para el software y para el hardware ha de ser una constante fuente de entradas bien sincronizadas, bien formateadas, y un perenne vigía receptor de salidas de todo tipo. A su vez, los ingenieros de sistemas esperan que los usuarios y demás ingenieros evitarán toda comunicación sistemática que no esté canalizada por medio del OS



Liz Dixon

Ésta suele ser el teclado, pero otras corrientes de entrada son la interface RS423 y el sistema actual de archivos, al cual se accede por medio de la orden *EXEC. La mayor rutina encargada de las entradas desde las citadas corrientes se llama OSRDCH (OS *Read*D CH*aracter*: lectura de caracteres del OS).

2. *Rutinas de salida y visualización*: Encargadas de las salidas generadas por el ordenador. En el BBC hay un buen número de corrientes de salida, desde la visualización en televisor hasta la impresora, la interface RS423 y el sistema de archivos manejado por medio de la orden *SPOOL. Pero además de encargarse de la impresión y de los demás modos de dar salida a los datos, estas rutinas del OS controlan el chip de visualización 6845 dentro del ordenador y el empleo de los caracteres definidos por el usuario, por citar tan sólo dos de las funciones extra atribuidas a estas rutinas. Las llamadas del OS son las siguientes, OSWRCH (OS WR*ite* CH*aracter*: escritura de caracteres del OS), OSASCII y OSNEWL.

3. *Sistemas de archivo*: Todo sistema operativo debe proporcionar al usuario los medios para guardar el contenido de la memoria del ordenador sobre algún soporte más duradero. La sección del OS que gestiona tales transacciones se llama *sistema de archivos actuales seleccionados*, y en el BBC se tienen las opciones de la cinta magnética, el disco, el Econet, la ROM o el telesoftware. Las rutinas de archivo del OS pueden servirse de otras rutinas adicionales en ROM que instruyan al OS para el trato con el hardware asociado con el sistema particular de archivo.

Para hacer posible la interconexión con medios de almacenamiento magnético a través de ROM paginadas para sistema de archivo, se incorporan un buen número de rutinas estándar del OS destinadas a la administración de archivos. Entre ellas se encuentran rutinas para escribir o leer archivos enteros, para la obtención o el envío de bytes individuales desde un fichero abierto y rutinas de lectura o escritura de grupos de bytes desde o hacia un archivo. Las siete llamadas del OS referentes a la gestión de archivos emplean cada una un vector para indicar la rutina adecuada en el sistema de almacenamiento en cassette. Si se conecta una ROM paginada de sistema de archivo, puede aunarse con programas existentes de gestión de archivos en BASIC o en assembly cambiando simplemente estos vectores para que apunten a sus propias rutinas en ROM. Una ROM de este tipo es la DFS ROM que permite al BBC Micro el empleo de unidades de disco flexibles.

4. *Interrupciones*: En esencia, una interrupción es una señal generada sea por el hardware sea por el software que indica a la CPU que interrumpa lo que actualmente está haciendo y realice la tarea que pide inmediata atención. Una vez realizada ésta, la CPU reanuda lo que estaba haciendo como si nada hubiera ocurrido. En el BBC existen numerosas facilidades de tratamiento de interrupciones a las que el usuario puede acceder gracias al OS.

Además de estas cuatro áreas principales, existen dos llamadas del OS de capital importancia que controlan diversas funciones de la máquina. Se llaman OSBYTE y OSWORD (*word*: palabra), empleadas para controlar el chip de sonido, la tecla de interrupción o *break* y cosas similares.

¿Por qué emplear llamadas?

En la mayoría de los casos es posible obtener, a los pocos meses del lanzamiento de una máquina, la información necesaria sobre la disposición interna de la memoria del ordenador y del hardware en general (salvo implicaciones legales). Si conocemos tales detalles ¿a qué viene interesarnos por las llamadas del OS? ¿No es más sencillo acceder a los dispositivos o a la memoria directamente?

En parte ya hemos dado respuesta a esta pregunta anteriormente: las llamadas del OS nos previenen contra futuros cambios del hardware o de la configuración llevados a cabo por el fabricante. De igual modo, si se llamara a las rutinas por medio de sus actuales direcciones en la ROM nos encontraríamos con dificultades en el momento en que esta ROM fuera modificada; si se accede a las rutinas de la ROM por medio de las adecuadas llamadas del sistema operativo, se tienen siempre previstas estas eventualidades.

La instrucción que citamos ahora escribirá, en un BBC Micro común, el valor 200 en la puerta para el usuario que está en la dirección &FE60.

```
?&FE60=200
```

Si se añade un segundo procesador al ordenador, esta rutina no enviará tal valor a la puerta para el usuario, sino que lo escribirá en una posición de memoria de este segundo procesador. Por medio de una adecuada llamada del OS podemos obviar la dificultad; la llamada sabrá cómo escribir el valor en la puerta para el usuario esté o no esté conectado un segundo procesador. Tal llamada sería:

```
*FX 151,96,200
```

Se trata de una de las muchas llamadas del sistema operativo que pueden emplearse para acceder a las áreas del BBC, tales como el VIA (*versatile interface adaptor*: interface adaptador versátil para el usuario) y el bus de 1 MHz.

Por otra parte, no hay que estar siempre inventando la rueda. Si ya existe dentro de la máquina una rutina para realizar una determinada función, ¿para qué vamos a preocuparnos de ir directamente a las rutinas de la ROM? El caso es que la llamada del sistema operativo es más eficaz que el acceso directo a las rutinas implicadas en dicha llamada.

Las únicas razones realmente convincentes para acceder directamente a la memoria o a los dispositivos del hardware sería si fuera necesaria una mayor rapidez en el acceso o si no existiera la llamada del OS que realizara la tarea requerida. Si la velocidad fuera un factor crucial, el acceso directo es con frecuencia más rápido que el camino por las rutinas del OS. Sin embargo, resulta bastante peliagudo el acceso directo a la máquina, y nos guardaremos mucho de hacerlo, recordando siempre que los programas que funcionan en una máquina determinada no sirven en otras máquinas con un OS o un hardware conectado diferente. Este tipo de problemas se encuentra más en el OS del BBC, debido a los numerosos cambios que ha sufrido desde su primera aparición, que en el Spectrum, que siempre ha permanecido fiel al mismo OS. La próxima lección estará dedicada al examen de las rutinas del OS del BBC.

Más adelante trataremos del OS del Spectrum y de otros microordenadores.

Récords olímpicos

"Summer games" (Juegos de verano), original juego escrito para el Commodore 64, incluye ocho pruebas deportivas de gran emoción, desde atletismo hasta gimnasia

Summer games (Juegos de verano), de Epyx, incluye ocho pruebas deportivas, desde atletismo y natación hasta gimnasia y tiro. En cada prueba pueden participar hasta ocho competidores distintos, quienes pueden elegir un país al cual representar entre un total de 17. Los ciudadanos de países cuyas banderas no estén incluidas tienen la oportunidad de participar bajo el estandarte de Epyx.

Una vez cargado el juego, la primera escena muestra la ceremonia de apertura. Acompañado por una música convenientemente heráldica, un atleta portador de la antorcha olímpica sube corriendo hasta una plataforma y enciende la llama, mientras se libera al cielo una bandada de palomas. El sonido y los gráficos de la ceremonia de apertura son una buena muestra del conjunto del paquete. El fondo está dibujado cuidadosamente utilizando gráficos de alta resolución trazados por mapa de bits, y los uniformes movimientos del corredor y de las palomas son resultado de las excelentes facilidades de sprites del Commodore. La música, si bien no es el mejor ejemplo de lo que se puede conseguir empleando el chip SID (*sound interface device*: dispositivo interface de sonido) del Commodore 64, hace buen uso de los tres osciladores disponibles. La impresión general es que el programa, no obstante, explota al máximo las capacidades del ordenador.

Los jugadores pueden optar por participar en todas las competencias, en una sola, practicar un deporte o ver una lista de los récords mundiales de cada deporte. Hay, asimismo, una opción que permite el empleo de una o dos palancas de mando, las cuales ofrecen a los jugadores la posibilidad de competir directamente entre sí en las pruebas de natación y carrera sin necesidad de recurrir a un temporizador de paso por ordenador.

La primera de las ocho pruebas es el salto con pértiga, que quizá sea la más difícil de todas. Los jugadores compiten de uno en uno respondiendo a una serie de preguntas por parte del ordenador. Cuando está preparada la altura mínima de la barra, de 4 m, se le pregunta al jugador si desea competir a esa altura. Si la respuesta es afirmativa, el ordenador pide entonces la posición de salida de la pértiga y el atleta empieza entonces a correr a lo largo de la pantalla.

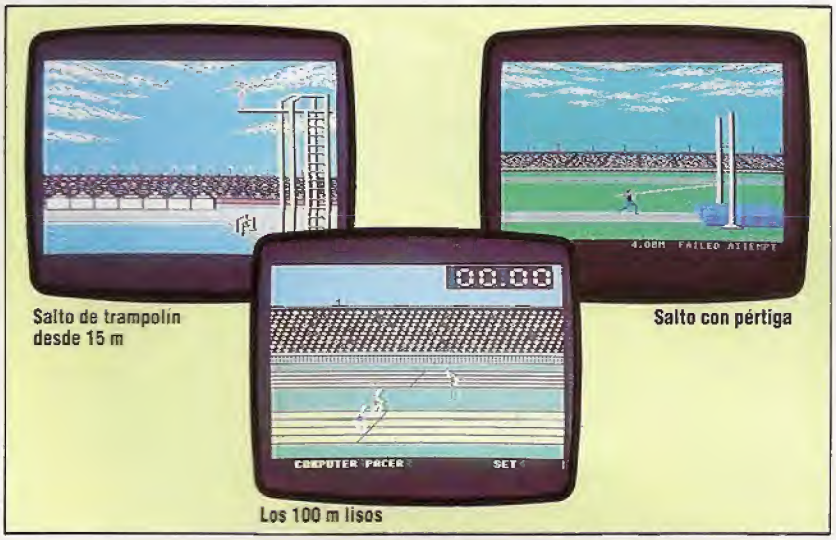
El jugador ha de tirar la palanca hacia atrás para fijar la pértiga, empujarla hacia adelante para levantar al atleta por encima de la barra y después accionar el pulsador de disparo para dejar caer la pértiga e impedir que la misma caiga contra la barra. Cada una de estas acciones exige una sincronización de fracción de segundo, puesto que un error de cálculo en cualquier punto provoca inmediatamente la caída de la barra.

Esta secuencia es una buena ilustración de la clase de parámetros que se han de considerar durante la codificación. No sólo es necesario que la pantalla esté totalmente apoyada en todo momento para asegurar un desplazamiento uniforme de los gráficos, sino que también el ordenador ha de verificar los movimientos producidos en la palanca de mando y el pulsador de disparo. Por último, el ordenador debe comprobar que la pértiga y el atleta estén en el ángulo y la posición correctos para saltar con éxito. Mientras se lleva a cabo todo ello, el usuario no debe notar que estas acciones están teniendo lugar.

Tras cada prueba aparece una tabla que muestra las medallas concedidas y se interpreta el himno nacional del país ganador. Entonces el ordenador carga la siguiente competición. El hecho de que cada prueba se cargue por separado es un indicador de la cantidad de código que se requiere para ejecutar cada prueba.

Las competiciones que siguen a continuación son salto de trampolín y gimnasia. Aquí el jugador debe manipular la palanca de mando para producir una zambullida o un salto uniforme, asegurándose de acabar con una suave entrada en el agua o de caer cuidadosamente sobre la colchoneta. El ordenador puntúa, entonces, la ejecución.

Compitiendo
La calidad del *Summer games* de Epyx se aprecia claramente en estas fotografías. Cada prueba se carga individualmente desde disco o cinta, lo que permite almacenar y procesar grandes cantidades de datos de alta resolución para los fondos



Salto de trampolín desde 15 m

Salto con pértiga

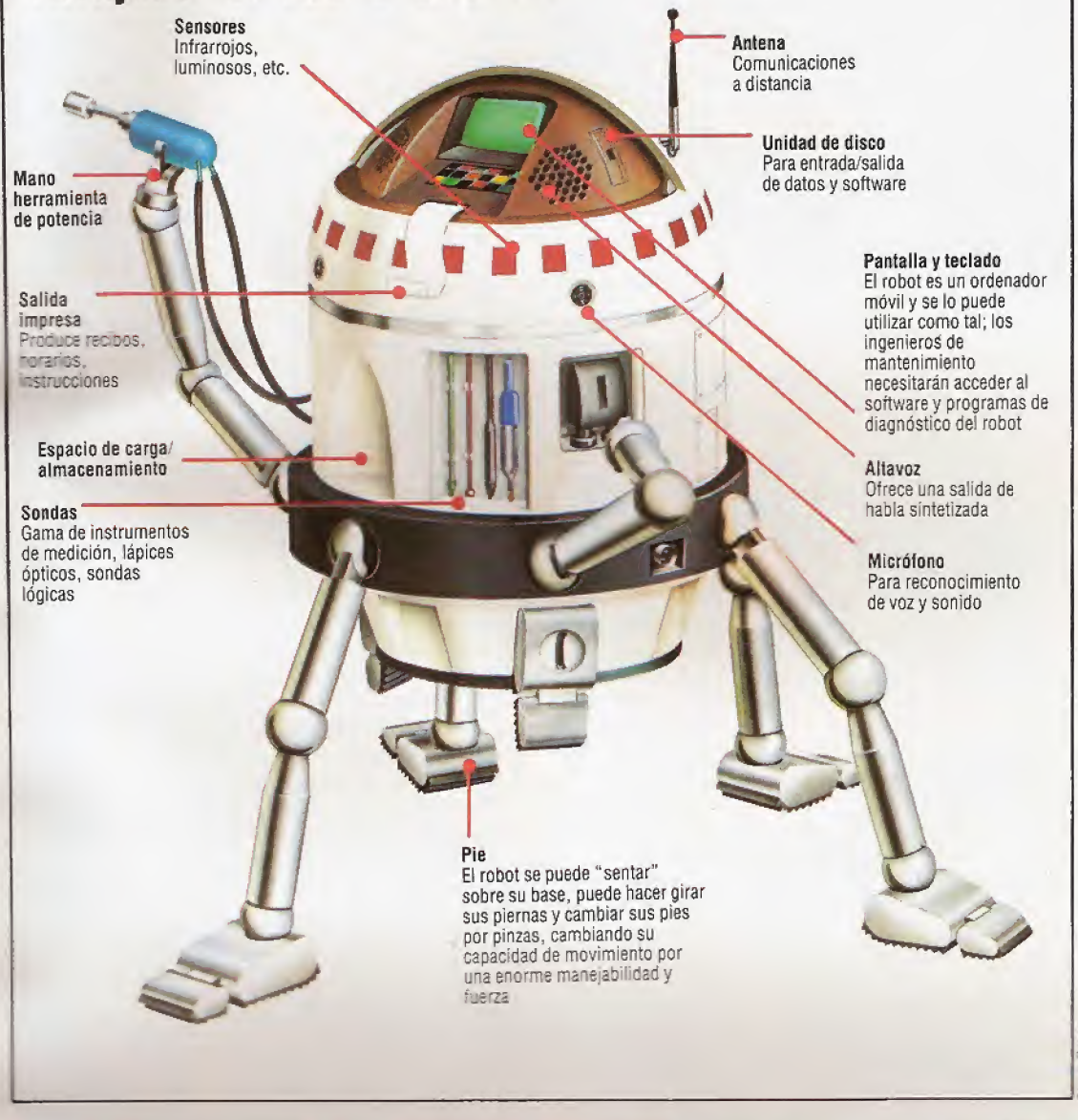
Los 100 m lisos

Ian McKinnell

Summer games: Para el Commodore 64.
Editado por: Epyx Software, 1043 Kiel Court, Sunnyvale, California, Estados Unidos.
Autores: Randy Glover, Stephen Landrum, John Leupp, Brian McGhie, Stephen Mundry, Erin Murphy, Scott Nelson
Palanca de mando: Necesaria
Formato: Versiones gemelas en cassette o disco

Lo que vendrá

Trabajador de cuello metálico



Steve Gross

Operario del mañana
Si alguna vez se desarrollara un robot con fines generales parecido al ser humano, a un costo que lo convirtiera en un sustituto razonable de la mano de obra semiespecializada, necesitaría una "inteligencia" sumamente desarrollada que comprendiera base de datos de conocimientos, integración sensorial, base de datos de habilidades y software para aprendizaje. Una inteligencia como ésta se podría empaquetar en una gran diversidad de cuerpos. He aquí uno de los posibles tipos, que podría funcionar como un obrero semiespecializado de la industria ligera o pesada

Finalizaremos esta serie dedicada a la robótica tratando de vislumbrar los futuros adelantos en este campo

Nuestra serie sobre robótica ha dejado claro cómo el mundo real de los robots continúa muy alejado del concepto que ha creado la literatura novelesca acerca de los seres pensantes mecánicos. Nuestra imaginación nos ha conducido a esperar ciertas cosas de los robots. Esperamos que sean capaces de moverse con entera libertad, alimentados con su propia energía; que vean, oigan y perciban el mundo que los rodea; que hablen con nosotros sobre ciencia y filosofía o que, al menos, se comuniquen de una forma inteligente, y que manejen objetos e ideas tal como lo hacemos nosotros. En otras palabras, en nuestra imaginación hemos creado los robots a nuestra propia imagen y semejanza. Cuando observamos con ojo crítico los robots comerciales, industriales y para aficionados existentes en la actualidad, a menudo nos sorprendemos de lo bien que pueden realizar sus tareas específicas, sintiéndonos, al mismo tiempo, algo decepcionados por el hecho de que no puedan hacer más.

Sabiendo lo que ya sabemos ahora acerca de la naturaleza del diseño y la implementación de ro-

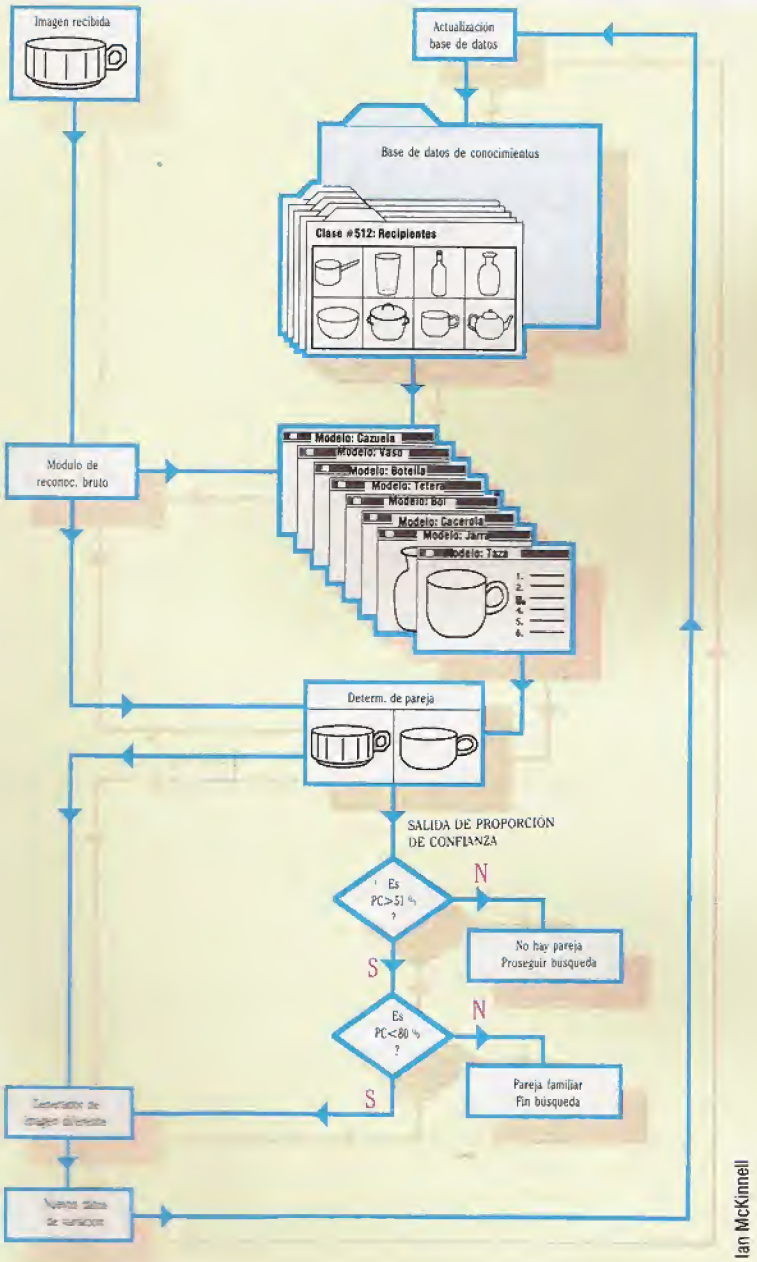
Variaciones visuales

El conocimiento acerca de un objeto se puede almacenar como una imagen "modelo" del objeto arquetípico, más una serie de sentencias de datos de variación; cada sentencia de variación se puede aplicar al modelo para producir una imagen diferente que aún se ajuste a la definición del tipo. Una imagen captada por los sensores del robot se explora mediante un módulo de análisis bruto que proporciona una primera hipótesis respecto a la clase de objeto que está a la vista. Cada objeto se compara con la imagen recibida hasta hallar una pareja. La confianza estadística con la cual se efectúe este emparejamiento determina si las variaciones de la imagen exigen la generación de una nueva sentencia de datos de variación

bots, ¿qué podemos esperar, realista y prácticamente, de los robots del futuro?

Movimiento

Es sumamente improbable que en un futuro cercano los robots caminen apoyados en algo que se parezca a una extremidad humana. Sería preciso destinar demasiado espacio de memoria y tiempo de proceso al esfuerzo que supone mantener el equilibrio, mientras que las junturas y la musculatura eléctrica o hidráulica carecen de la flexibilidad o la libertad de que gozan los seres humanos gracias a la interacción de músculos, tendones y cartílagos. Además, existen muchas ocasiones en las que el robot se vería muy limitado por el hecho de tener que andar sobre dos piernas. No obstante, recientemente se han construido algunos robots experimentales con 4 o 6 extremidades, con apariencia de insectos. Éstos podrían ofrecer una interesante variación de diseño para algunas aplicaciones robot.



Ian McKinnell

Para otras aplicaciones, como pueden ser las operaciones militares, la exploración sideral y muchos usos convencionales en el hogar, las ruedas suelen proporcionar el método más práctico de movimiento, y es poco probable que esta situación se modifique. El desplazamiento del robot se volverá más fluido, pero probablemente jamás pueda competir con la belleza y la armonía de un atleta humano en movimiento.

Muchos robots industriales y brazos-robot más pequeños están necesariamente fijos en un sitio, con sus movimientos restringidos a un área de acción muy específica, dado que están diseñados para llevar a cabo una o dos tareas bien definidas. A menos que la naturaleza del montaje industrial cambie de forma radical, es probable que incluso los robots industriales ambulantes continúen relativamente confinados: seguirán siguiendo huellas, desplazándose sobre raíles o colgando de guías elevadas. Es posible que los avances en el campo de la automatización y el diseño robótico den lugar a un cambio radical en los métodos de producción industrial, pero es imposible predecir en qué consistirá.

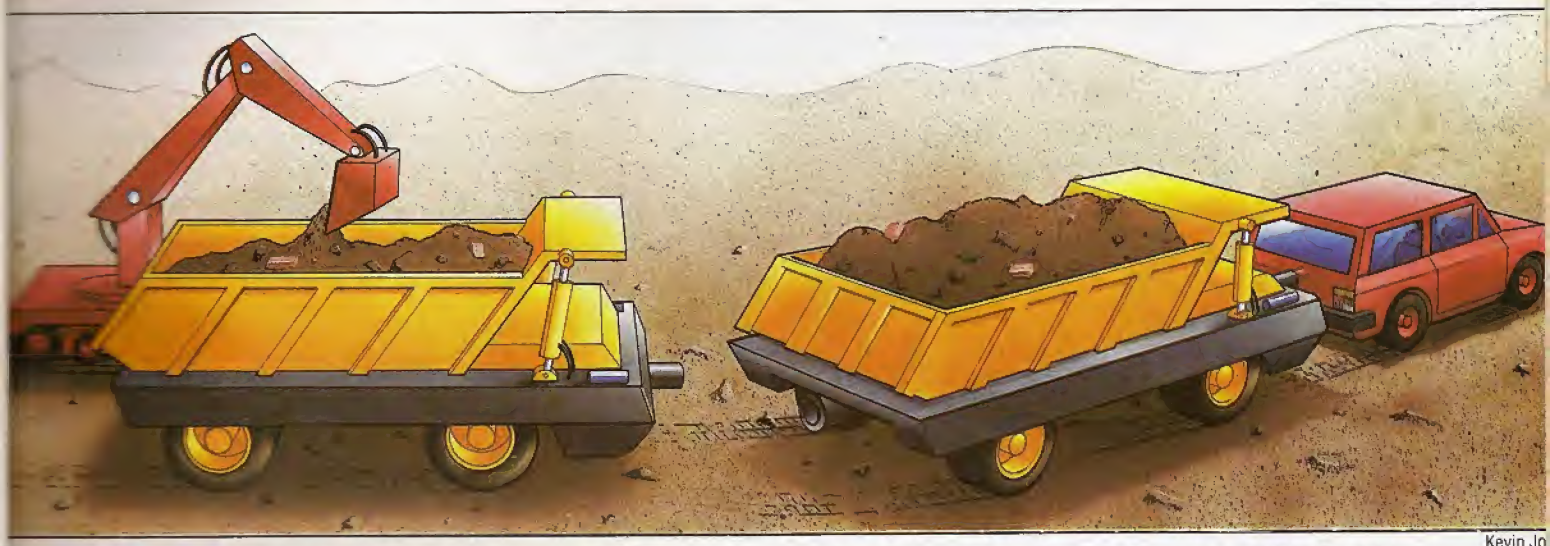
Un elemento clave del movimiento es la necesidad del robot de responder a su entorno. Esto significa que el robot debe estar bien equipado con un sistema sensorial y que la entrada sensorial debe estar en relación directa con su movimiento.

Sensores

Se puede dotar a los robots de equipos sensoriales sofisticados que amplíen sus percepciones a áreas nuevas o desconocidas para los humanos. Los sensores de proximidad, los detectores de movimiento, los dispositivos de realimentación posicional discreta y precisa y los detectores de ruido con una gama muy amplia de frecuencias perceptibles, le confieren a un robot la capacidad de captar datos más variados que los que puede recoger un ser humano. Los sistemas visuales se están volviendo más exactos, con un aumento de la resolución de las imágenes percibidas. Las técnicas de síntesis y reconocimiento de voz serán, ciertamente, cada vez más sofisticadas y desempeñarán un importante papel en el desarrollo de la robótica.

Es muy probable que los robots de utilidades, empleados principalmente para aplicaciones industriales, continúen estando dotados sólo de aquellos sensores necesarios para llevar a cabo las tareas que se les han asignado. Los brazos-robot soldadores, por ejemplo, no tienen ninguna necesidad de disponer de voz ni de una compleja realimentación visual. Pueden llevar a cabo sus trabajos con precisión y rapidez con un mínimo de entrada sensorial, y las percepciones externas posiblemente constituirían más bien un obstáculo.

Los robots para fines generales, diseñados para aprender a partir de la experiencia e imitar los procesos del pensamiento humano, habrán de estar equipados con la mayor cantidad de sensores posible. Sería crucial que el robot fuera capaz de investigar su entorno independientemente y asimilar la información recopilada. Los seres humanos dependen en gran medida, por ejemplo, de una combinación de realimentación visual y auditiva para comprender el habla. Nosotros a menudo tendemos a ignorar esta síntesis de sensaciones, en particular cuando pensamos en términos del diseño de robots.



Kevin Jo

Convoy de anticipación
Es probable que las técnicas de la robótica produzcan su mayor impacto en la sociedad cuando se las incorpore a herramientas de grado inferior para fines especiales tales como grúas, máquinas excavadoras, vehículos de reparto local y transporte pesado. En la ilustración vemos una máquina excavadora robot cargando un tren de camiones robot en una obra en construcción. Cuando está cargado, cada camión avanza semiinteligentemente hasta un punto de ensamblaje y se engancha por sí mismo a un tren de camiones. El tren circula por las carreteras gracias a la energía de los camiones individuales que lo componen, pero controlado por el conductor humano del vehículo guía. La combinación de la capacidad humana de decisión y dirección con la fuerza bruta y la primaria inteligencia de los robots probablemente redundará en una utilización más racional y rentable de los recursos existentes y de la tecnología del futuro

Pero para que un robot se comunique con un ser humano, para entender el lenguaje en vez de tan sólo reconocerlo, esta combinación de vista y sonido resulta esencial.

En la actualidad, la cantidad de datos que un robot puede aceptar y manejar está severamente restringida por la cantidad de memoria necesaria para almacenar la entrada sensorial, y por las limitaciones de la potencia y la velocidad de proceso. Un robot puede almacenar una imagen visual de un objeto, como puede ser una manzana, y relacionar la imagen con un nombre. El almacenamiento de la imagen ocupa memoria y, cuanto mejor sea la resolución visual del robot, más memoria se necesitará para guardar la imagen. Puesto que, además, todas las manzanas no son iguales, el robot debe tener memoria suficiente para almacenar un muestreo característico de imágenes de manzanas, o bien un algoritmo que reconozca las variaciones y sea capaz de hacer girar la imagen básica de modo que la manzana se pueda contemplar desde cualquier posición. Aun con un mínimo nivel de resolución (p. ej., 256 pixels por imagen), la cantidad de variaciones puede muy bien ser de varios millares.

En el futuro es probable que las demandas de memoria se satisfagan mediante chips de RAM de mayor capacidad (actualmente se están desarrollando chips de 1 Mbit) y a través de la utilización de chips de RAM exclusivos que almacenen datos "de variaciones". El procesador puede ir llamando a los datos de fines generales retenidos en estos chips a medida que sea necesario para clarificar diversas imágenes diferentes.

Además de una enorme memoria, la auténtica conciencia sensorial de un robot requeriría que los datos entraran simultáneamente desde muchas fuentes y que pudieran ser procesados rápidamente. Los procesadores existentes serían incapaces de tratar todo el volumen de información que entraría en cualquier momento dado, y enseguida comenzarían a amontonarse datos en espera de ser procesados. Una solución muy probable para este problema es el empleo de dos o más procesadores de gran velocidad y capacidad trabajando en paralelo. Un procesador controlador actuaría entonces como administrador, distribuyendo tareas entre los procesadores del sistema que estuvieran desocupados.

El progreso hacia la resolución de los problemas de hardware con que se encuentran los investigadores se está produciendo a gran velocidad. Pero un robot necesitará de un software muy complejo que le permita comprender lo que se está procesando. Es decir, el robot necesita una mente para saber lo que tiene que hacer con sus percepciones.

La mente

Tal como hemos visto al analizar el movimiento y los sensores, hay dos direcciones principales para la robótica. La primera, y la que cuenta con mayores probabilidades de explotarse pronto, es el área de las herramientas inteligentes, o robots de utilidades. Los brazos industriales y los sistemas de fabricación automatizada que llevan a cabo tareas específicas, independientemente de lo complejas que éstas sean, sólo necesitan estar provistos de un software controlador cuidadosamente definido. A un brazo robot se le puede dar un juego de coordenadas y programarlo para que ejecute una secuencia de acciones sin comprender lo que está haciendo, dónde se encuentra, ni ningún otro detalle relativo a su entorno. El resultado podría ser una puerta perfectamente pintada en una cadena de montaje de coches, o un coche muy bien lavado en una cadena automatizada de lavado de automóviles. Sin embargo, en la medida en que se les solicite a los robots la realización de una mayor diversidad de tareas, éstas estarán necesariamente definidas con menos claridad. Si tienen que desplazarse por una habitación en la cual el contenido cambia día a día, deben ser capaces no sólo de reunir y procesar información, sino también de incorporar nuevas percepciones en su comprensión del mundo. Al robot se le debe proporcionar software operativo y controlador, pero es preciso que haya lugar para que éste se incremente.

El tratar de crear una mente mecánica plantea importantes cuestiones, que de momento aún están sin resolver, acerca de la forma en que piensan y aprenden los humanos. Por ejemplo, ¿qué sabe una criatura en el momento de nacer? El humano ya adulto, ¿es enteramente producto de su entorno o de su herencia, y cuál es la relación entre éstos? ¿Parte el humano con un conjunto de construcciones internas que lo ayudan a aprender lengua, matemáticas, etc.?; de ser así, ¿cómo funcionan? Es difícil responder a estas preguntas sin poder experimentar en el cerebro humano.

Poder mental

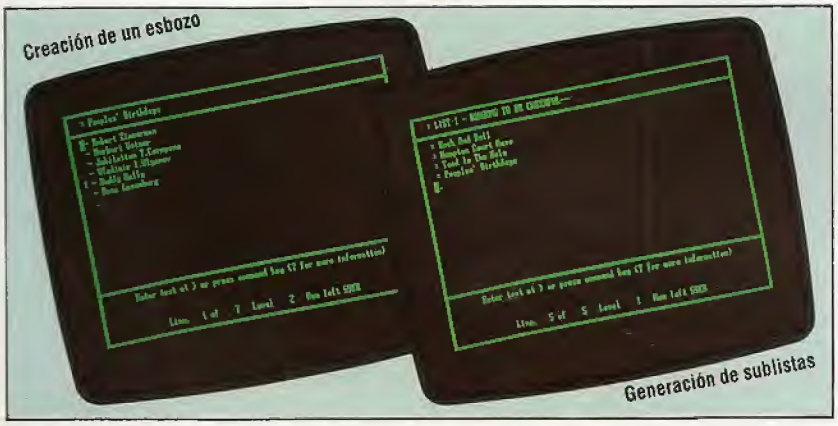
Al "BrainStorm", avanzado paquete de software vertical, se le considera el primer "procesador de pensamientos"

El *BrainStorm* ha sido calificado como un "procesador de pensamientos", pero esto no significa que el programa intente emular al cerebro. Lo que hace realmente el *BrainStorm* (literalmente, "idea genial") es ayudar al usuario a organizar sus pensamientos. Será útil una analogía con los métodos preelectrónicos para comprender qué hace.

Cuando se planifica cualquier proyecto complejo, resulta útil confeccionar listas de lo que se tiene que hacer. Una lista de objetivos principales generará sublistas de cómo completar cada punto de la lista precedente, y así sucesivamente hasta que el planificador queda totalmente mareado de tanto andar revolviendo trozos de papel. Cualquier cambio (p. ej., decidir que uno de los puntos de la lista principal en realidad debe incluirse en la sublista de otro punto) puede significar tantos borrones y enmiendas que el sistema se vuelve inmanejable. *BrainStorm* hace que esta clase de cambios y desarrollos sean tan sencillos como las funciones de "recortar y pegar" de que disponen los procesadores de textos; de modo que, al fin y al cabo, la expresión "procesador de pensamientos" tampoco está tan errada.

Cualquier elemento de cada lista o sublista puede ser *promovido*, convirtiéndose en el título de una sublista inferior; y, si es necesario enlazar entre sí actividades similares de listas diferentes, se les puede dar el mismo nombre, aludiendo a ellos como *tocayos*. En ese caso, todo lo que se añada a una sublista encabezada por un tocayo le será añadido también a las otras listas que utilizan el tocayo.

Quienes estén familiarizados con las más bien hostiles facilidades de edición de un paquete para tratamiento de textos como el *WordStar*, encontrarán que es muy fácil acostumbrarse a las instrucciones para control del cursor no mnemotécnicas. Pulsando la tecla Control (CTRL) conjuntamente con la tecla S, el cursor se mueve hacia la izquierda. CTRL–D, CTRL–E y CTRL–X mueven el cursor hacia la derecha, arriba y abajo, respectivamente, y estas cuatro teclas forman en el teclado un patrón lógico. De todos modos, estas teclas se pueden redefinir. A lo que sí resulta menos sencillo habituarse es al hecho de que uno puede desplazar el cursor hacia arriba y hacia abajo mientras está entrando texto, pero tiene que pasar a la modalidad "corrección" (utilizando CTRL–A) para desplazarse hacia la izquierda o la derecha de una línea.



Operación del programa

El menú de apertura del *BrainStorm* ofrece 11 opciones, cada una de las cuales se selecciona mediante una letra inicial; la mayoría de éstas se explican por sí solas: U*se (usar)*, Load (cargar), Print (imprimir), ID Drive (para pasar a una unidad diferente de la unidad por defecto), Clear (limpiar, para borrar de la memoria el modelo actual), Save (guardar), Write (escribir, para grabar en disco), Directory (directorio), Xit, Merge (mezclar) y Kill.

Para empezar, el usuario pulsa U y el programa entra inmediatamente en modalidad de teclado. Se pueden entrar ideas como listas más o menos aleatorias (lo que se conoce como un *model*: modelo). P. ej., un modelo podría estar compuesto por:

Leer manual
Comenzar a teclear lista
Teclear sublista
Editar lista

Las instrucciones de control no aparecen de forma inmediata, pero se listarán pulsando ?. Desplazando el cursor hacia arriba hasta cualquiera de los elementos de la lista y pulsando CTRL–R, se "promueve" el elemento designado, el cual pasa a ser el encabezamiento de una sublista. Del mismo modo, cualquier elemento de esa sublista se puede convertir en el encabezamiento de una nueva sublista, y así sucesivamente hasta agotar la memoria. Para retornar a la lista anterior, se pulsa CTRL–C.

Los elementos se pueden trasladar, ya sea dentro de una misma lista como a listas de otros niveles, etiquetándolos con el signo @ y utilizando luego CTRL–G (de Get: tomar) o CTRL–P (de Put: poner) para ejecutar el movimiento. Después de emplear CTRL–G, el signo @ pasa automáticamente al siguiente elemento de la lista; por consiguiente, volviendo a pulsar CTRL–G se pasará el elemento siguiente, y así sucesivamente. Ésta es una valiosa facilidad para trasladar una serie completa de elementos a una sublista inferior o superior.

Si se desea insertar nuevos elementos en una lista ya existente, sólo hay que desplazar el cursor hasta el comienzo de la línea que ha de ir tras el elemento nuevo, y entrarlo. Cuando se pulsa RETURN el resto de la lista se desplaza una línea hacia abajo para hacerle sitio. Pulsando CTRL–A (de Amend: corregir) se puede modificar cualquier elemento.

Al darle el mismo nombre (p. ej., una fecha) a elementos de listas diferentes, éstos se convierten en *tocayos* y automáticamente se establece entre ellos una referencia cruzada. De modo que si una lista dada exigiera que determinado acontecimiento sucediera en un día especial, supongamos el 1 de enero, se podría crear una lista de acontecimientos para esa fecha y ello permitiría actualizar continuamente un horario de las actividades del día mediante la referencia cruzada.

Se puede crear cualquier cantidad de tocayos y acceder después a ellos en secuencia, utilizando CTRL–S para la ocurrencia siguiente y CTRL–D para la previa. Esto se conoce como una lista doblemente encadenada circular, de modo que al final de la última ocurrencia CTRL–S retornará a la primera.

Las listas, por supuesto, se pueden imprimir y, por lo general, se las formatea con indentaciones que representan los niveles de la lista. Por ejemplo:

```
Leer manual
  Sacar el manual de la caja
    Buscar índice
      Hallar página requerida
    Leer página
  Volver a poner el manual en la caja
Empezar a entrar lista
  Pulsar CTRL – R para promover el elemento de la
  lista como un nuevo encabezamiento
    Digitar sublista
  Pulsar CTRL – C para retornar a la lista anterior
Editar lista
  Pulsar CTRL – A para modificar una entrada
```

El tamaño del indentado lo especifica el usuario.

También se pueden editar listas desde fuera del *BrainStorm*. Si han sido salvadas utilizando la instrucción *Write-to-disk* (grabar en disco), el *WordStar* y otros programas para tratamiento de textos las reconocerán como documentos, y se las podrá editar, imprimir y, de ser necesario, volver a guardar. Esto significa, por ejemplo, que se puede utilizar el *BrainStorm* para preparar la sinopsis de un libro, que puede ser luego escrito utilizando un programa normal para tratamiento de textos, accediendo al archivo BrainStorm.Doc a medida que avanza el libro.

Gracias a la flexibilidad del programa, este proceso puede comenzar a partir de los primeros apuntes e ideas generales (nuevamente, algo que por lo general se realiza en trozos de papel) que después se convertirán en títulos de capítulos (bajo los cuales se listan los contenidos). Si un elemento de un capítulo se vuelve lo suficientemente extenso como para convertirse en un capítulo separado, esto se puede realizar con suma facilidad. Del mismo modo, se pueden "degradar" títulos de capítulos que resulten ser menos importantes de lo que se pensó en un primer momento.

De esta forma, el esbozo general del libro empieza a surgir y, dado que el *WordStar* o programas similares pueden acceder a archivos del *Brain Storm*, la transformación de este esbozo en el manuscrito acabado va fluyendo naturalmente a partir de los primeros procesos intelectuales. Esto significa que no es necesario adoptar los procedimientos, bastante laboriosos, del *BrainStorm*, si uno desea, por ejemplo, imprimir a dos o tres espacios.

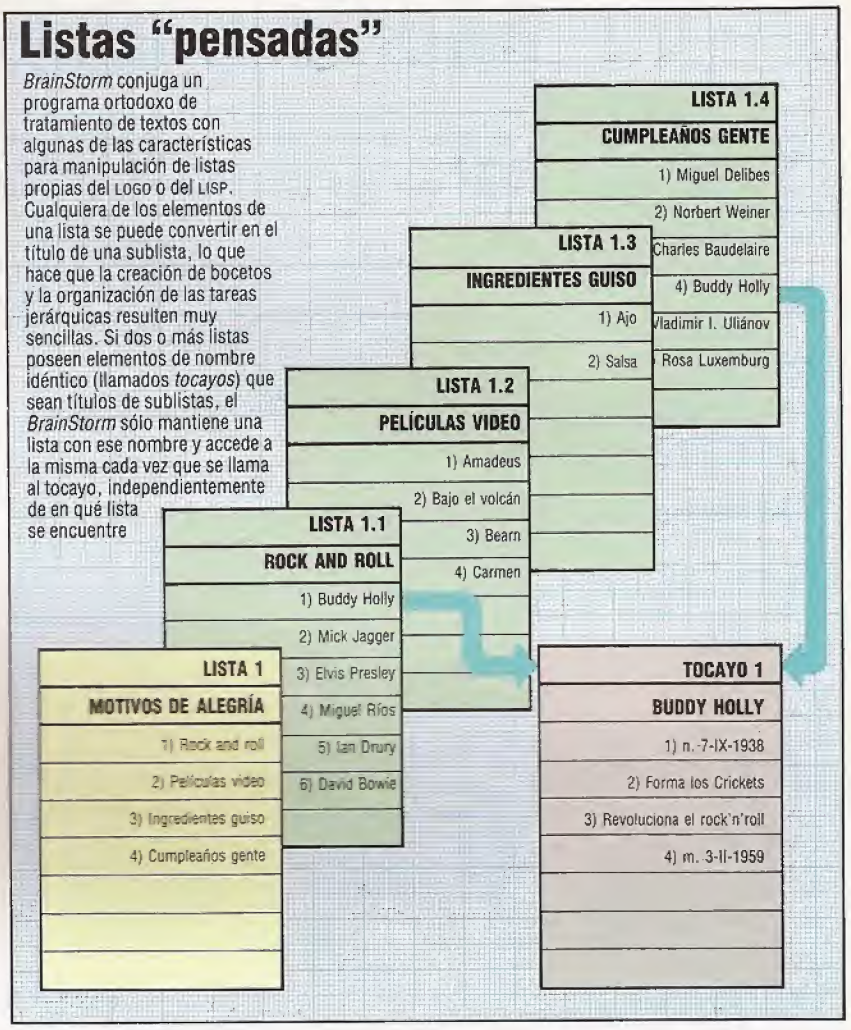
El programa viene con un modelo de muestra (llamado SAMPLE.BRN), que incluye el esqueleto de un planificador de horarios, archivo de nombres y direcciones, una lista de tareas (distribuidas en sublistas de "urgentes", "importantes" y "no olvidar"), más una sección para notas. Este modelo es bastante valioso y puede ser añadido a los modelos propios mediante el empleo de la opción *Merge*.

Aprender a usar el *BrainStorm* es fácil. El manual de hojas sueltas es muy claro (fue escrito utilizando el *BrainStorm*), pero las instrucciones siempre están a disposición del usuario en el menú de la pantalla, por lo que la referencia continua al manual resulta innecesaria.

No todas las instrucciones son mnemotécnicas, por lo que algunas podrían resultarle difíciles de recordar al usuario; pero se proporciona un programa INSTALLB, que permite volver a configurar todas las instrucciones y alterar los menús para adaptarlos a la nueva estructuración. Este programa es muy claro y totalmente activado por menú.

Listas "pensadas"

BrainStorm conjuga un programa ortodoxo de tratamiento de textos con algunas de las características para manipulación de listas propias del LOGO o del LISP. Cualquiera de los elementos de una lista se puede convertir en el título de una sublista, lo que hace que la creación de bocetos y la organización de las tareas jerárquicas resulten muy sencillas. Si dos o más listas poseen elementos de nombre idéntico (llamados *tocayos*) que sean títulos de sublistas, el *BrainStorm* sólo mantiene una lista con ese nombre y accede a la misma cada vez que se llama al tocayo, independientemente de en qué lista se encuentre



BrainStorm: Para unas 25 máquinas CP/M, MS-DOS y PC-DOS, incluyendo IBM, Sirius y Apricot
Editado por: Caxton Software Ltd., 10-14 Bedford Street, London WC2E 9HE, Gran Bretaña
Autores: David Tebbutt y Mike Liardet
Formato: Disco

Misión especial

Prosiguiendo con el proyecto, desarrollaremos las rutinas para dejar objetos y nos ocuparemos de los escenarios especiales

La subrutina DEJAR tiene muchas similitudes con la rutina RECOGER que describimos en el capítulo anterior. De hecho, podemos utilizar las mismas rutinas de comprobación de objetos que se desarrollaron para utilizar con la instrucción RECOGER. En la rutina RECOGER se llevan a cabo tres comprobaciones del objeto. La primera está diseñada para comprobar si la segunda parte de la sentencia de la instrucción contiene o no un objeto válido. Esto se realiza comparando sistemáticamente cada palabra de la frase de la instrucción con los nombres de los objetos de la matriz del inventario, IV$(,). De hallar una pareja, se establece una variable, F, que da la posición del objeto emparejado dentro de la matriz. Esta comprobación de validez también se debe emplear en la rutina DEJAR para comprobar si el objeto existe y, de ser así, determinar su posición en el inventario.

La segunda comprobación utilizada en la rutina RECOGER también se emplea en la rutina DEJAR; ésta comprueba si el jugador lleva el objeto especificado en la instrucción en el inventario de objetos transportados, IC$(). Obviamente, ¡el jugador no puede dejar un objeto que no esté llevando consigo! La tercera prueba utilizada en la rutina RECOGER efectúa una comprobación para asegurar que el objeto a recoger esté en el escenario en curso del jugador, determinado por P, la variable de posición. Sin embargo, dado que el objeto a dejar debe estar en posesión del jugador, su posición no aparecerá en el inventario principal y, por consiguiente, esta tercera prueba no es necesaria en la rutina DEJAR.

Suponiendo que ambas comparaciones produzcan un resultado favorable, se deben introducir las siguientes modificaciones tanto en el inventario principal como en el del jugador:

1) La posición del objeto a dejar se especificará ahora mediante F. La posición actual, P, debe ahora entrarse en la matriz del inventario principal en la posición IV$(F,2).
2) La descripción del objeto debe eliminarse del inventario personal de los objetos que lleva consigo el jugador, IC$(). La mejor forma de hacerlo es buscar en la matriz hasta hallar el objeto apropiado y sustituirlo por una serie nula.

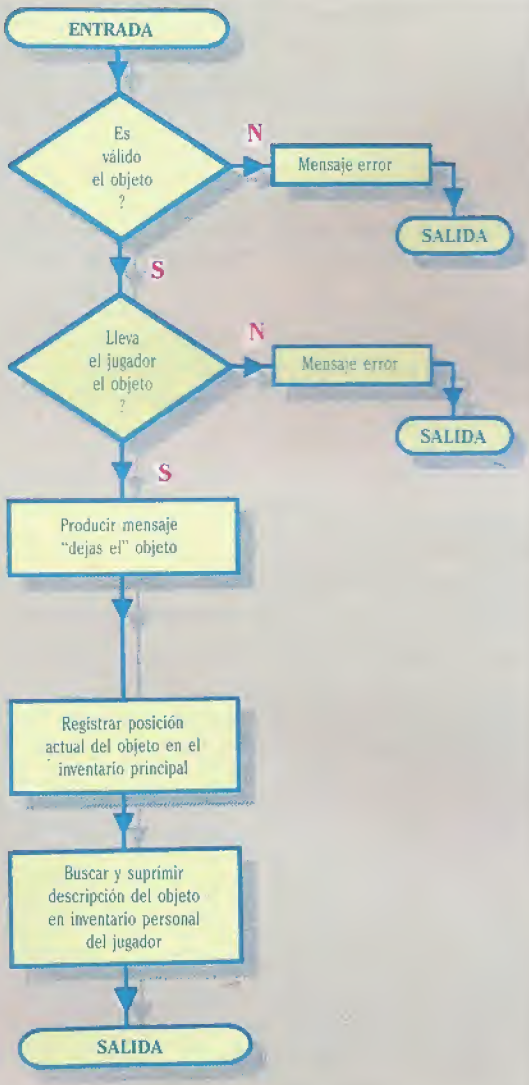
En el diagrama de flujo vemos la lógica de la rutina DEJAR. Éste es el listado de la rutina para el juego *El bosque encantado*:

```
3900 REM **** S/R DEJAR ****
3910 GOSUB 5300:REM OBJETO VALIDO
3920 IF F=0 THEN SN$="NO HAY NINGUN "+W$:GOSUB 5500:RETURN
3930 :
3940 REM ** ESTA OBJETO EN INVENTARIO OBJETOS TRANSPORTADOS
     **
3950 OV=F:GOSUB5450
3960 IF HF=0 THEN SN$="TU NO TIENES EL "+IV$(F,1):
     GOSUB5500:RETURN
3970 :
3980 REM ** DEJAR OBJETO **
3990 SN$="DEJAS EL "+IV$(F,1):GOSUB5500
4000 IV$(F,2)=STR$(P):REM EFECTUAR ENTRADA EN EL INVENTARIO
4010 :
4020 REM ** ELIMINAR OBJETO DEL INVENTARIO DE TRANSPORTADOS
     **
4030 FOR J=1TO2
4040 IF IC$(J)=IV$(F,1) THEN IC$(J)=" ":J=2
4050 NEXT J
4060 RETURN
```

Podemos ver que una de las principales ventajas de programar por módulos es que se puede acceder a las mismas rutinas para fines diferentes. Utilizando un sistema de banderas o indicadores, las decisiones se pueden tomar dentro de breves subrutinas sobre las cuales no se actúa hasta que el control retorne a la rutina que llamó a la subrutina. Un buen ejemplo del empleo de esta clase de estructura de programa es la prueba de validez que descri-

¡Suéltalo!

La lógica de la rutina DEJAR es similar a aquella de la rutina RECOGER, pero solamente la validez del objeto y su presencia en el inventario del jugador han de ser comprobados antes de ejecutar la instrucción DEJAR. El inventario principal es entonces actualizado para mostrar un nuevo escenario para el objeto, y el nombre del objeto es eliminado del inventario del jugador



Ian McKinnell

bimos anteriormente. Esta subrutina es llamada tanto por la rutina RECOGER como por la rutina DEJAR. En cada caso, la subrutina toma una decisión respecto a la validez del objeto incluido en la sentencia de la instrucción. Sin embargo, el flujo del programa no sufre ninguna alteración hasta que se produzca un RETURN a cualquiera de las dos rutinas, RECOGER o DEJAR. Sólo después del retorno la subrutina de la prueba de validez establece el valor de la bandera F, y se produce la bifurcación adecuada. La crítica que se puede hacer a esta técnica es que en realidad estamos comprobando dos veces la misma condición: una para establecer el valor de la bandera y otra para comprobar su valor. Aunque esto es efectivamente así, la mayor flexibilidad y facilidad para la depuración que se consigue mediante el empleo de esta técnica por lo general compensa la contrapartida resultante, es decir, el tiempo de ejecución ligeramente mayor.

Escenarios especiales

Hemos llegado a un punto de nuestro proyecto en el cual tenemos completa la programación del esqueleto del juego: es decir, la programación que permite que el jugador lleve objetos consigo y se desplace a través del mundo de aventuras. Podemos ahora pasar a la siguiente fase de diseño, en la cual consideramos los escenarios "especiales" donde se utilizan los objetos, acechan los peligros y se pone a prueba el ingenio y la destreza del jugador.

Antes de analizar con detalle la programación de las rutinas para uno de los escenarios especiales de *El bosque encantado*, consideremos el código que debemos introducir en el bucle principal del programa para detectar los escenarios especiales. En el listado se han de insertar estas dos líneas:

```
257 GOSUB2700:REM ES ESPECIAL P?
258 IF SF=1 THEN 300:REM SIGUIENTE INSTRUCCION
```

La línea 257 llama a un subrutina para ver si el escenario en curso es especial. De ser así, entonces se pone a uno una "bandera especial", SF. Esto significa que cuando el control retorne finalmente al bucle principal del programa, se podrá evitar la parte del bucle principal que se ocupa de las instrucciones. La subrutina que decide si el escenario actual es especial o no es ésta:

```
2700 REM **** S/R ES ESPECIAL P ****
2705 SF=0:REM RESTAURAR BANDERA ESPECIAL

2710 REM ** OTROS ESCENARIOS ESPECIALES **
2720 ON P GOSUB4590,4590,4790,4590
2730 RETURN
```

Recordará que, cuando diseñamos el mapa original para *El bosque encantado*, numeramos primero los cuatro escenarios especiales. Por consiguiente, podemos simplificar la selección de la subrutina apropiada para cada escenario especial haciendo uso de la instrucción ON...GOSUB. Tal como se ve a partir de la forma en que se la utiliza en la línea 2720, esta instrucción va seguida por una serie de números de línea, y el número de línea apropiado se selecciona de acuerdo al valor de la variable de posición, P. Si, por ejemplo, P es uno, la instrucción bifurcará (GOSUB) a la subrutina que comienza en el primer número de línea de la lista; si P es dos, para la llamada GOSUB se utilizará el segundo número de línea, y así sucesivamente.

Hay cuatro números de línea, uno para cada uno de los escenarios especiales de *El bosque encantado*. Si P fuera mayor de cuatro, entonces el control pasaría simplemente a la línea siguiente. Si cada una de las cuatro subrutinas a las que se puede llamar desde la línea 2720 establece una bandera, SF, se puede detectar el hecho de que P era un escenario especial.

De no llamarse a ninguna rutina, la bandera SF permanecerá establecida en cero, indicando que P no es más que un escenario normal. La instrucción ON...GOSUB es, evidentemente, una alternativa económica a la serie de sentencias IF...THEN que prueban el valor de una variable y bifurcan consecuentemente hacia subrutinas diferentes.

La entrada al túnel

Dos de los escenarios especiales de *El bosque encantado* son las dos entradas a un túnel (escenarios 1 y 4). Para tratar con la simple posibilidad de que el jugador desea entrar en el túnel, necesitamos construir cuidadosamente una rutina que manipule las instrucciones normales y le permita al jugador entrar en el túnel o dar la vuelta y volver al sendero.

```
4590 REM **** S/R ENTRADA AL TUNEL ****
4600 SF=1
4605 SN$="HAS LLEGADO A LA BOCA DE UN LARGO
     TUNEL":GOSUB5500
4610 SN$="PUEDES ENTRAR EN EL TUNEL O RETROCEDER POR EL
     SENDERO":GOSUB5500
4620 :
4625 PRINT:INPUT"INSTRUCCIONES";IS$
4630 GOSUB2500:REM DESCOMPONER INSTRUCCION
4635 IF F=0 THEN 4625:REM INSTRUCCION NO VALIDA
4637 GOSUB3000:REM INSTRUCCIONES NORMALES
4640 IF MF=1 THEN RETURN:REM EL JUGADOR RETROCEDE
4645 IF VF=1 THEN 4625:REM INSTRUCCION OBEDECIDA
4650 REM ** NUEVAS INSTRUCCIONES **
4655 IF VB$="ENTRAR" THEN GOSUB 4700:RETURN
4660 IF VB$="RETROCEDER" AND P=4 THEN MF=1:P=6:RETURN
4665 IF VB$="RETROCEDER" AND P=1 THEN MF=1:P=9:RETURN
4667 SN$="NO COMPRENDO":GOSUB5500:GOTO 4625
```

La rutina empieza por poner SF a uno para indicar el hecho de que se ha llegado a un escenario especial. Después de visualizar un mensaje en la pantalla, de describir la entrada al túnel y las opciones de que dispone el jugador, se solicita una instrucción. Nuevamente, en vez de reinventar la rueda cada vez que deseamos analizar una instrucción, podemos sacar partido de la construcción modular del programa para llamar a las subrutinas "descomponer instrucción" e "instrucción normal" desarrolladas para utilizar en las rutinas RECOGER y DEJAR. Considerando cuidadosamente los estados de las diversas banderas establecidas por estas dos subrutinas, podemos transferir el control al interior de nuestra nueva rutina según se requiera. Analicemos estas banderas de forma individual.

La bandera F establecida por la rutina "descomponer instrucción" indica si la instrucción que se le ha pasado tiene un formato válido. Si la instrucción es de una sola palabra no reconocida por la rutina, entonces F toma el valor cero, en cuyo caso querremos saltar hacia atrás del bucle para tomar otra instrucción.

La bandera MF es establecida por la rutina "instrucción normal" si se requiere la descripción de un escenario; esto sucede cuando se genera una instrucción AVANZAR o MIRAR. Un RETURN al bucle principal del programa permitirá el desplazamiento hasta un nuevo escenario en el primer caso, o que

se describa el mismo escenario y se vuelva a entrar a la rutina especial, en el segundo caso.

La rutina "instrucción normal" establece, asimismo, la bandera VF. Un valor de uno indica que se ha reconocido y obedecido la instrucción, en cuyo caso saltaríamos hacia atrás del bucle para la siguiente instrucción. Si VF <> 1, la instrucción no es ninguna de las normales. Habiéndonos ocupado de las posibilidades de instrucciones normales, podemos añadirle nuevas instrucciones a esta rutina. En este caso, se incluyen dos: ENTRAR, para penetrar en el túnel, y RETROCEDER, para alejarse un escenario desde la entrada al túnel. Puesto que esta rutina está diseñada para funcionar para las dos entradas al túnel, la instrucción RETROCEDER ha de considerar cuál de las dos entradas al túnel está salvando el jugador; ello lo indica P según sea su valor 1 o 4. Por lo tanto, P se puede poner a cero antes de dejar la rutina, de modo que al volver a entrar en el bucle principal del programa se produce un cambio de escenario.

Listados de "Digitaya"

```
1190 GOSUB2670:REM ES ESPECIAL P
1200 IF SF=1 THEN 1250:REM SIGUIENTE BUCLE

2360 REM ** S/R DEJAR **
2370 GOSUB5730:REM ES VALIDO EL OBJETO
2380 IF F=0 THEN PRINT"NO HAY NINGUN";W$:RETURN
2390 :
2400 REM ** LLEVA EL JUGADOR EL OBJETO? **
2410 OV=F:GOSUB5830
2420 IFHF=0THENPRINT"TU NO TIENES EL";IV$(F,1):RETURN
2430 :
2440 REM ** DEJAR EL OBJETO **
2450 SN$="DEJAS EL"+IV$(F,1):GOSUB5880
2460 IV$(F,2)=STR$(P):REM ACTUALIZAR POSICION OBJETO
2470 :
2480 REM ** SUPRIMIR EN LISTA OBJETOS TRANSPORTADOS **
2490 FORJ=1TO4
2500 IF IC$(J)=IV$(F,1)THENIC$(J)=" ":J=4
2510 NEXT J
2520 RETURN

2670 REM **** S/R ES ESPECIAL P ****
2680 SF=0:REM RESTAURAR BANDERA ESPECIAL

2710 ON P GOSUB 2850,2960,3450,3830,4180,4550,5150
2720 RETURN

2850 REM **** S/R TOMA TV ****
2860 SF=1
2870 SN$="HAS ENTRADO EN LA TOMA DEL TELEVISOR Y NO TIENES
     ESCAPATORIA."
2880 SN$=SN$+"ESTAS CONDENADO PARA SIEMPRE A SER UN INVITADO
     DE UN PROGRAMA DE TERTULIA DE LA TV"
2890 GOSUB5880:REM FORMATEAR IMPRESION
2900 PRINT
2910 PRINT"BIENVENIDO AL PROGRAMA....."
2920 FORJ=1TO500:NEXTJ
2930 GOTO 2910
2940 END

3830 REM **** PUERTA PALANCA MANDOS ****
3840 SF=1
3850 SN$="UN USUARIO CON LOS OJOS ENROJECIDOS DISPARA
     REPETIDAMENTE SU LASER CONTRA TI."
3860 GOSUB5880:REM FORMATO
3870 :
3880 REM ** INSTRUCCIONES **
3890 RD=RDN(TI):IF RD>.65THEN 4110:REM ACERTADO
3900 PRINT:INPUT"INSTRUCCIONES";ISS
3910 GOSUB 1700:GOSUB 1900:REM ANALIZAR INSTRUCCION
3920 IFMF=1THENMF=0:PRINT"NO PUEDES
     MOVERTE...TODAVIA":GOTO3880
3930 IFVF=1THEN3880:REM SIGUIENTE INSTRUCCION
3940 IF VB$<>"USAR" THEN PRINT"NO COMPRENDO":GOTO3880
3950 GOSUB5730:REM ES VALIDO EL OBJETO
3960 IFF=0THENPRINT"NO HAY NINGUN";NN$:GOTO3880:REM SIGUIENTE
     INSTRUCCION
3970 :
3980 REM ** ES EL OBJETO ESCUDO LASER **
3990 IF F=3 THEN4020:REM OK
4000 SN$="TU "+IV$(F,1)+"NO SIRVE":GOSUB5880:GOTO3880
4010 :
4020 OV=3:GOSUB5830:REM TIENE CONSIGO EL ESCUDO LASER
4030 IFHF=0THENSN$="TU NO TIENES EL
     "+IV$(3,1):GOSUB5880:GOTO3880
4040 :
4050 REM ** SALVADO **
4060 SN$="UTILIZAS EL ESCUDO LASER PARA PROTEGERTE. UNA RAFAGA
     TE ARROJA"
4070 SN$=SN$+"FUERA DE LA PUERTA PARA PALANCA DE MANDOS Y TE
     HACE ENTRAR OTRA VEZ EN LA MAQUINA."
4080 GOSUB5880:REM FORMATO
4090 P=INT(RND(TI)*40+7):MF=1:RETURN
4100 :
4110 REM ** ACERTADO **
4120 SN$="HAS SIDO ALCANZADO POR EL LASER Y APENAS SI ERES
     CONSCIENTE DE QUE"
4130 SN$=SN$+"TUS ATOMOS HAN QUEDADO DISEMINADOS POR LAS
     CUATRO ESQUINAS"
4140 SN$=SN$+" DEL UNIVERSO"
4150 GOSUB5880:REM FORMATO
4160 END

5150 REM **** S/R PUERTA A LA MEMORIA ****
5160 SF=1
5170 SN$="ERES RECIBIDO POR UN PORTERO QUE TE DICE QUE NO PUEDES
     SER ADMITIDO"
5180 SN$=SN$+"A MENOS QUE DES UNA DIRECCION":
     GOSUB5880
5190 REM ** INSTRUCCIONES **
5200 PRINT:INPUT "INSTRUCCIONES";ISS
5210 GOSUB1700:GOSUB1900:REM ANALIZAR
5220 IF MF=1 THEN RETURN:REM IRSE
5230 IF VF=1 THEN 5200:REM SIGUIENTE INSTRUCCION
5240 IF VB$<>"DAR"THENPRINT"NO COMPRENDO":
     GOTO 5200
5250 :
5260 GOSUB5730:REM ES VALIDO EL OBJETO
5270 IFF=0THENPRINT"NO HAY NINGUN";W$:GOTO5200:REM SIGUIENTE
     INSTRUCCION
5280 :
5290 REM ** ES EL OBJETO UNA DIRECCION **
5300 IF F=1 THEN5330:REM OK
5310 PRINT"EL NECESITA TU DIRECCION":GOTO5200
5320 :
5330 OV=1:GOSUB5830:REM LLEVA CONSIGO LA
     DIRECCION
5340 IF HF=1 THEN 5370
5350 SN$="NO TIENES LA"+IV$(1,1):GOSUB5880:
     GOTO5200
5360 :
5370 REM ** OK PUEDE PASAR **
5380 SN$="EL PORTERO EXAMINA TU DIRECCION Y TE
     PERMITE"
5390 SN$=SN$+"ENTRAR":GOSUB5880
5400 P=40:MF=1:RETURN
```

Complementos al BASIC

Spectrum:

En ambos programas reemplace los siguientes nombres de variables: SN$ por S$, NN$ por R$, IV$(,) por V$(,), IC$ por I$(), IS$ por T$, VB$ por B$.

En el listado de *El bosque encantado* sustituya las siguientes líneas:

```
2720 IF P=1 THEN GOSUB4590
2722 IF P=2 THEN GOSUB4690
2724 IF P=3 THEN GOSUB4790
2726 IF P=4 THEN GOSUB4590
```

En los listados para *Digitaya* reemplace estas líneas:

```
2710 IF P=1 THEN GOSUB2850
2711 IF P=2 THEN GOSUB2960
2712 IF P=3 THEN GOSUB3450
2713 IF P=4 THEN GOSUB3830
2714 IF P=5 THEN GOSUB4180
2715 IF P=6 THEN GOSUB4550
2716 IF P=7 THEN GOSUB5150
3890 LET RD=RND(1)
4090 LET P=INT(RND(1)*40+7)
```

BBC Micro:

En los listados de *Digitaya* reemplace estas líneas:

```
3890 RD=RND(1)
4090 P=RND(40)+7
```

La prueba del tiempo

Examinemos una de las primeras máquinas portátiles compatibles con IBM que aparecieron en el mercado: el Compaq Plus

El lanzamiento en Estados Unidos del IBM Personal Computer confirió cierta respetabilidad al microordenador. Los hombres de negocios, que hasta entonces habían considerado tales dispositivos como poco más que artilugios novedosos, se sintieron felices de convertirse en clientes del célebre gigante de la industria de ordenadores. Dado que las firmas de software comenzaron a crear un ingente número de programas para el IBM, fue inevitable que muchos fabricantes de hardware produjeran ordenadores capaces de ejecutar el software IBM y aprovecharan de este modo la inmensa base de software que se estaba acumulando.

Una de las pioneras en este campo fue la Compaq Computer Corporation, empresa creada específicamente para producir una máquina portátil compatible con IBM: el Compaq Plus. El modelo que vamos a examinar aquí es la versión de 256 Kbytes con unidades de disco gemelas, si bien existen versiones de la máquina con una sola unidad o con un disco rígido fijo de 10 Mbytes.

Chris Stevens

"Compaqtando"
Las líneas suaves y bien definidas de la carcasa del Compaq se repiten en el sencillo panel frontal y en el moderado facsímil del teclado del IBM-PC. A ambos lados de la carcasa, unas tapas dan acceso a las puertas y a la fuente de alimentación eléctrica, como asimismo a un espacio de almacenamiento para el cable de potencia y el enchufe de la red, ¡un punto importante y en ocasiones menospreciado!

Teclado y pantalla

"Portátil" tal vez sea un término erróneo para una máquina que pesa 14 kg; ciertamente es difícil transportarla más allá de una corta distancia. Compaq ha reconocido el problema que supone trasladar un peso tan grande y ha proporcionado un asa vinílica acolchada que facilita el transporte.

Como es habitual tratándose de ordenadores portátiles, el teclado del Compaq se engancha en la parte delantera del ordenador. Dado que el asa para transportarlo está instalada en la parte trasera, ello significa que el teclado sirve asimismo como base. No obstante, la máquina es de construcción sólida y el teclado parece aceptar el peso y el consiguiente traqueteo bastante satisfactoriamente. La unidad completa tiene un tamaño aproximado al de una máquina de coser. Una vez desenganchado, el teclado se conecta al ordenador mediante un cable en espiral. El fabricante afirma que esto permite situar el teclado de manera que se pueda hallar la posición de trabajo más cómoda. Sin embargo, si se separa de la unidad más de unas ocho pulgadas, comienza a ser estirado por la espiral del cable, limitando por lo tanto la distancia entre la pantalla y el usuario. Esto se podría haber solucionado con un cable más delgado.

El teclado, al igual que el propio ordenador, posee patas plegables que permiten colocarlo en ángulo para trabajar con comodidad. Tal como uno esperaría de un ordenador de oficina como el Compaq Plus, las teclas son fiables y fáciles de usar. La disposición del teclado es idéntica a la del IBM-PC, con 10 teclas de función junto al lado izquierdo de las teclas de máquina de escribir, y un teclado numérico en el lado derecho. La ventaja obvia de esta disposición es que los usuarios que estén habituados a la del IBM podrán utilizar el teclado del Compaq instantáneamente sin tener que aprenderse la nueva posición de las teclas. La desventaja reside en que el teclado del IBM no es el ideal y, por consiguiente, se reproducen los mismos problemas de diseño. La tecla Enter del IBM es del mismo tamaño que las otras teclas, por lo cual hallarla resulta difícil. Lo que quizá sea más grave para quienes escriben al tacto es que la tecla Shift no está en el lugar habitual, es decir, debajo de la tecla A. Esta posición la ocupa, en cambio, la barra invertida, lo que significa que el mecanógrafo al tacto se pasará, al menos en principio, pulsando continuamente la barra invertida en vez de Shift. Estos problemas se reproducen en el Compaq.

Cuando se ejecuta el MS-DOS (la versión Microsoft del PC-DOS), las teclas de función situadas a la izquierda del teclado actúan como medios auxiliares a la edición. Estas funciones varían a tenor del programa de aplicaciones que se esté ejecutando. Con el BASIC, por ejemplo, se convierten en teclas de entrada de palabras clave individuales, tales como LOAD, SAVE y LIST. Las máquinas MSX utilizan un sistema similar para las teclas de función.

El teclado numérico situado a la derecha del teclado tiene una doble función. En operación normal, se lo puede emplear para desplazar el cursor a través de la pantalla, pero si se pulsa la tecla Num Lock se lo puede utilizar a modo de calculadora.

La pantalla es un monitor verde estándar de 17,7 × 13,3 cm, con una resolución de 80 × 25 caracteres. El texto se lee fácilmente y a la derecha de la pantalla hay un control de brillo. En el lado izquierdo están las unidades gemelas de disco flexible de 5 1/4 pulgadas. Debido a que la configuración estándar del Compaq Plus es una única unidad de disco, el disco maestro del sistema MS-DOS dará por sentado que es éste el caso y pedirá que todos los discos se coloquen en la unidad A; esto puede ser una molestia cuando se está copiando uno, porque el usuario tiene que intercambiar los discos continuamente. Sin embargo, el MS-DOS se puede configurar para hacer un uso completo de las dos unidades. Las unidades propiamente dichas parecen ser más silenciosas que sus equivalentes IBM.

Conexiones al exterior

Debajo de un panel al lado derecho de la máquina están las puertas de interface. El Compaq Plus está equipado con una puerta para impresora en paralelo tipo Centronics, una interface RGB y una puerta RF. También se proporcionan tres ranuras de ampliación, permitiendo instalar placas con conectores compatibles con IBM. Las ampliaciones típicas son placas de memoria extra, una tarjeta de color VDU o un modem que le permita al ordenador comunicarse a través de la red telefónica. Debajo de otro panel situado en el lado izquierdo está la entrada de potencia (un Eurosocket estándar de tres patillas), encima de la cual está el interruptor on/off. A uno de los lados está el ventilador que mantiene frío el interior de la máquina. Dentro del ordenador, el sistema de circuitos está protegido por una carcasa metálica (motivo, en parte, del peso de la máquina); ésta no sólo ofrece protección contra manejos bruscos, sino que también hace de escudo contra las interferencias de radio que pudieran perturbar el proceso. El metal actúa asimismo como disipador.

El ordenador viene acompañado de tres manuales: una guía de funcionamiento y guías de referencia de MS-DOS y de BASIC. Insólitamente, tratándose de este tipo de máquina, las guías están encuadernadas en forma de libro y no son las usuales carpetas de anillas. Los libros vienen junto con los discos del sistema en carpetas plásticas imitación de ante. De los tres manuales, sólo la guía para operadores tiene un enfoque didáctico. De modo que quien desee documentarse sobre los diversos pasos de la utilización del BASIC o del MS-DOS hará mejor en adquirir otros textos de enseñanza.

Como una de las más antiguas máquinas compatibles con IBM, el Compaq, al igual que el propio IBM-PC, utiliza el procesador Intel 8088 en vez del chip 8086 más avanzado que emplean algunos de sus competidores más recientes, como el Olivetti M25. Si bien el 8088 es un procesador de 16 bits con un bus de direcciones de 20 bits, posee un bus de datos de 8 bits, a diferencia del 8086, que posee un bus de 16 bits. Esto significa que el Compaq es mucho más lento para recuperar datos que sus rivales que poseen el 8086, aunque, por supuesto, opera a la misma velocidad que el PC.

Respecto a la cuestión, sumamente importante, de la compatibilidad con el software IBM, el Compaq Plus se encuentra en una posición muy airosa. Hasta un programa tan difícil como el *Lotus 1-2-3* se puede ejecutar en la máquina. Sin embargo, quizá este hecho no resulte sorprendente si se tiene en cuenta que el presidente de Compaq también es miembro de la junta directiva de Lotus Software. Una de las pocas cosas que el Compaq Plus no puede ejecutar es el disco de diagnóstico IBM; pero, puesto que ese programa interroga directamente a la ROM del sistema de E/S de BASIC (BIOS: BASIC Input-Output System), ello tampoco ocasiona inconvenientes.

El Compaq Plus es, en definitiva, una de las máquinas compatibles con IBM más fiables que existen en el mercado, con una trayectoria ya probada.

En el aspecto negativo, la máquina está empezando a dar muestras de su edad, y no sólo por el chip 8088, ya anticuado. Está empezando a dar la sensación de que están contados los días de los micros "transportables" de más de 10 kg. Con el creciente y continuo perfeccionamiento de los ordenadores "de regazo", la aparición de una máquina de este tipo compatible con el IBM es sólo cuestión de tiempo.

Costo-prestaciones

Los principales argumentos de venta del Compaq Plus son su compatibilidad con IBM y, por consiguiente, el acceso a la enorme base de software IBM. Aquí se puede apreciar el costo de un sistema IBM-PC equiparable a las especificaciones básicas del Compaq Plus:

Configuración	Compaq Plus	IBM-PC
Precio básico	£ 2 524	£ 1 310
Discos gemelos 360 K	estándar	341
Pantalla color alta res.	estándar*	300
Tarjeta salida monitor	estándar	208
Tarjeta gráficos color	estándar	223
128 K de RAM	estándar	86
Sistema operativo y BASIC basado en disco	estándar	61
Teclado	estándar	213
Total	2 524	2 742

* La pantalla incorporada es monocromática de alta resolución, pero el Compaq posee una salida en color RGB como estándar.

Chris Stevens

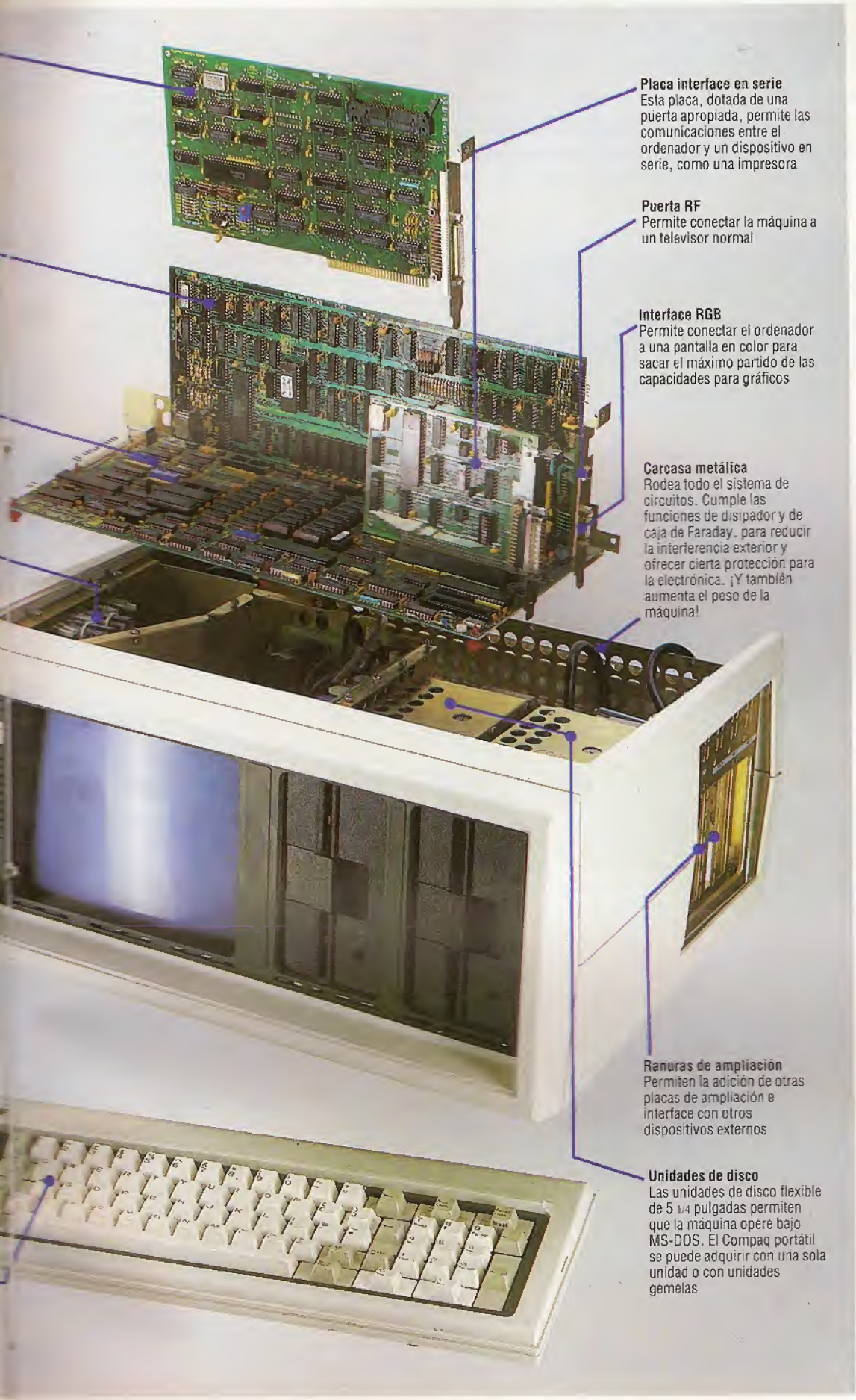
Placa impresora
Esta placa con una interface para impresora en paralelo permite conectar la máquina a una impresora de tipo Centronics

Placa VDU
Se encaja en una de las puertas para ampliación y proporciona el sistema de circuitos necesario para activar un televisor o pantalla externa

Placa de circuitos principal
Contiene los 256 K de RAM, el procesador 8088 y los chips de entrada/salida. Asimismo hay disponibles ranuras libres para chips extras tales como el procesador de matemáticas 8087

Fuente de alimentación eléctrica
Está instalada aquí, con un ventilador por detrás para mantener refrigerado el interior de la máquina

Teclado estilo IBM
Es idéntico a la versión IBM. Observe las teclas de función a la izquierda y el teclado numérico a la derecha



Placa interface en serie
Esta placa, dotada de una puerta apropiada, permite las comunicaciones entre el ordenador y un dispositivo en serie, como una impresora

Puerta RF
Permite conectar la máquina a un televisor normal

Interface RGB
Permite conectar el ordenador a una pantalla en color para sacar el máximo partido de las capacidades para gráficos

Carcasa metálica
Rodea todo el sistema de circuitos. Cumple las funciones de disipador y de caja de Faraday, para reducir la interferencia exterior y ofrecer cierta protección para la electrónica. ¡Y también aumenta el peso de la máquina!

Ranuras de ampliación
Permiten la adición de otras placas de ampliación e interface con otros dispositivos externos

Unidades de disco
Las unidades de disco flexible de 5 1/4 pulgadas permiten que la máquina opere bajo MS-DOS. El Compaq portátil se puede adquirir con una sola unidad o con unidades gemelas

Chris Stevens

COMPAQ PLUS

DIMENSIONES
480 x 400× 200 mm

CPU
Intel 8088, 4,7 MHz

MEMORIA
256 K de RAM, ampliables a 640 K

PANTALLA
Textos: 25 filas de 80 columnas. Gráficos: 640 x 200 pixels

INTERFACES
Impresora en paralelo tipo Centronics, RGB y un enchufe hembra RF. También hay ranuras de ampliación que permiten instalar otras placas de interface

LENGUAJES
BASIC cargado desde el disco maestro del sistema

TECLADO
47 teclas tipo máquina de escribir, 14 teclas de control, 10 teclas de función y 10 teclas de función para cálculo

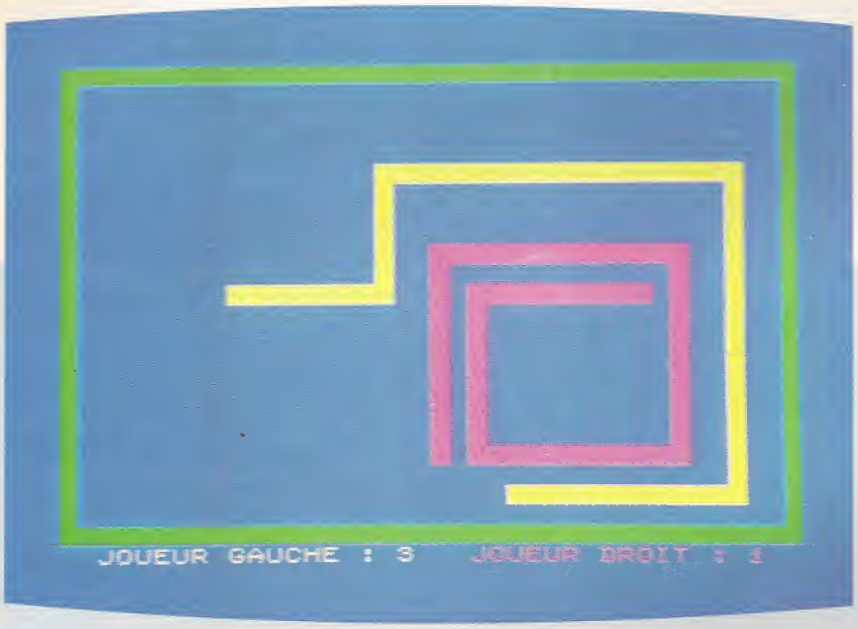
DOCUMENTACION
Se suministran tres manuales: una guía de funcionamiento, una guía de referencia del MS-DOS y una guía de referencia de BASIC. De ellos sólo la guía de funcionamiento lleva al lector paso a paso a través de los procedimientos. Los principiantes deberán adquirir otros manuales con fines didácticos

VENTAJAS
La construcción de la máquina es sólida y su compatibilidad con el IBM-PC está demostrada

DESVENTAJAS
El peso de la máquina la coloca más bien en la categoría de "transportable". Asimismo, el Compaq Plus está en peligro de ser superado por la tecnología más reciente

Trazos

Esta versión de este famoso juego de acción ha sido escrita para el MO5 de Thomson. También está disponible para el TO 7



Dos jugadores se enfrentan para dividirse el espacio vital. Cada uno de ellos debe esforzarse, mientras va desplazándose, por no cortar en ningún momento su trazado o el de su adversario ni salirse del rectángulo dibujado en la pantalla. Utilice las palancas de mando o las teclas siguientes:

Jugador de la derecha: P, L, M y @.

Jugador de la izquierda: Z, Q, S y W.

```
 10 REM **********
 20 REM * TRAZOS *
 30 REM **********
 40 CLEAR ,,1
 50 GOSUB 750
 60 ON JK GOTO 130
 70 D$=INKEY$
 80 HB=(D$="L")-(D$="M")
 90 VB=(D$="P")-(D$="@")
100 HA=(D$="Q")-(D$="S")
110 VA=(D$="Z")-(D$="W")
120 GOTO 170
130 HA=(STICK(0)=7)-(STICK(0)=3)
140 VA=(STICK(0)=1)-(STICK(0)=5)
150 HB=(STICK(1)=7)-(STICK(1)=3)
160 VB=(STICK(1)=1)-(STICK(1)=5)
170 IF HA<>0 THEN H1=HA:V1=0
180 IF VA<>0 THEN V1=VA:H1=0
190 IF HB<>0 THEN H2=HB:V2=0
200 IF VB<>0 THEN V2=VB:H2=0
210 X1=X1+H1
220 Y1=Y1+V1
230 IF SCREEN(X1,Y1)=127 THEN 350
240 LOCATE X1,Y1
250 COLOR 3
260 PRINT N$;
270 X2=X2+H2
280 Y2=Y2+V2
290 IF SCREEN(X2,Y2)=127 THEN 410
300 LOCATE X2,Y2
310 COLOR 5
320 PRINT N$;
330 BEEP
340 GOTO 60
350 F2=F2+1
360 GOSUB 690
370 IF F2=10 THEN 470
380 IF INKEY$<>" " THEN 380
390 GOSUB 870
400 GOTO 60
410 F1=F1+1
420 GOSUB 690
430 IF F1=10 THEN 540
440 IF INKEY$<>" " THEN 440
450 GOSUB 870
460 GOTO 60
470 CLS
480 COLOR 1,7
490 LOCATE 9,5
500 PRINT "GANA JUGADOR DERECHA";
510 LOCATE 15,10
520 PRINT F2;"A";F1;
530 GOTO 600
540 CLS
550 COLOR 1,7
560 LOCATE 9,5
570 PRINT "GANA JUGADOR IZQUIERDA";
580 LOCATE 15,10
590 PRINT F1;"A";F2;
600 LOCATE 14,15
610 PRINT "OTRA ?";
620 IF INKEY$<>" " THEN 620
630 D$=INKEY$
640 IF D$=" " THEN 630
650 IF D$<>"N" THEN RUN
660 CLS
670 SCREEN 4,6,6
680 END
690 FOR I=1 TO 5
700 BEEP
710 FOR J=1 TO 100
720 NEXT J
730 NEXT I
740 RETURN
750 CLS
760 SCREEN 2,4,4
770 DEFINT A-Z
780 ATTRB 1,1
790 LOCATE 6,10,0
800 PRINT "PALANCAS DE MANDO ?";
810 R$=INKEY$
820 IF R$=" " THEN 810
830 JK=-(R$="0")
840 ATTRB 0,0
850 DEFGR$(0)=255,255,255,255,255,255,255,255
860 N$=GR$(0)
870 CLS
880 COLOR 2
890 FOR X1=0 TO 39
900 LOCATE X1,0:PRINT N$;
910 LOCATE X1,23:PRINT N$;
920 NEXT X1
930 FOR Y1=1 TO 22
940 LOCATE 0,Y1:PRINT N$;
950 LOCATE 39,Y1:PRINT N$;
960 NEXT Y1
970 LOCATE 2,24:COLOR 3
980 PRINT "JUGADOR IZQUIERDA :";F1;
990 LOCATE 22,24:COLOR 5
1000 PRINT "JUGADOR DERECHA :";F2;
1010 X1=8:Y1=11:X2=32:Y2=11:H1=1:V1=0
1020 H2=-1:V2=0:S1=0:S2=0:RETURN
```

En primer plano

En esta ocasión estudiaremos distintos tipos de visualización de datos y desarrollaremos una rutina para crear gráficos de barras

Los gráficos de barras son un sencillo método gráfico de representar ciertos tipos de datos numéricos. El objetivo de un gráfico de barras es ayudar a entender a simple vista un conjunto de cifras sin tener que examinarlas con mayor detalle. Existen muchos programas para gráficos de gestión para trazar gráficos de barras, gráficos de tarta e histogramas, y estos programas a menudo están enlazados con hojas electrónicas de modo que se puedan visualizar los valores calculados en las mismas.

Nosotros, por el momento, no seremos tan ambiciosos, sino que comenzaremos por analizar cómo dibujar un gráfico de barras utilizando el LOGO. La versión del programa que vamos a considerar es para el Commodore 64; para ejecutar el programa en otras máquinas, el lector habrá de remitirse a los *Complementos al* LOGO, que incluyen los cambios necesarios en cada caso.

El proceso del dibujo de un gráfico de barras se divide en tres etapas:

a) tomar la entrada;
b) hallar el valor mayor (esto es necesario para que podamos colocar a escala las columnas de modo que quepan en la pantalla);
c) dibujar el gráfico de barras, a escala apropiada, y añadir luego las etiquetas.

El procedimiento de nivel superior se denomina GRAFICOBARRAS.INIC establece el valor de una cantidad de constantes que necesita el programa. Reuniéndolas todas en un mismo lugar se simplifica la labor de introducir modificaciones. COLORES contiene la lista de colores a utilizar para las columnas; si no le agrada nuestro esquema de color, ¡cámbielo! Tal como está, el programa imprimirá un máximo de 15 columnas, de modo que sólo se necesitarán 15 colores como máximo.

Entrada y cálculo

La entrada que se requiere consta de dos datos para cada una de las barras del gráfico; primero, el título a imprimir al pie de la barra y, segundo, la cantidad o el valor de la barra, o sea, su altura. En nuestras rutinas de entrada hemos incorporado algunas sencillas comprobaciones de verificación para asegurar que la entrada sea sensata. TOMAR.ENTRADA divide en dos partes la tarea de obtener la entrada, tomando los títulos para cada columna e insertando los valores correspondientes. Cuando acabe de entrar los datos, digite FIN como el siguiente "título".

TOMAR.TITULO acepta el título; rechazará una entrada en blanco pero aceptará cualquier otro valor. TOMAR.CANTIDAD aceptará como entrada sólo un número; cualquier otra entrada hará que el programa solicite que los datos sean reentrados. Cuando TOMAR.ENTRADA posee un par nombre/número válido, éste se añade al final de una lista de elementos de datos. Los elementos se deben entrar por el orden en el que uno desea que se tracen en la pantalla, de izquierda a derecha.

CALCULAR utiliza TOMAR.MAX para hallar el valor mayor y luego emplea éste para establecer una escala. Una regla común es que la altura del gráfico de barras debe ser de aproximadamente las tres cuartas partes de la anchura.

DIBUJAR.GRAFICO calcula la anchura de cada barra del gráfico, dibuja un eje hacia arriba de la pantalla, colorea y etiqueta las barras.

La anchura se calcula como un múltiplo de ocho. Ello se hace con el fin de evitar algunos problemas originados por la forma en que el Commodore 64 visualiza los colores, porque en la modalidad para gráficos normal del LOGO no se pueden tener dos colores en el mismo bloque de 8×8 pixels. DIBUJAR.EJE dibuja la línea hacia arriba de la pantalla y marca en ella el mayor de los valores de los datos; esto nos proporciona una forma sencilla de estimar los valores de las columnas.

Debemos retroceder desde la línea del eje antes de imprimir este número junto a la máscara que indica el valor mayor. El espacio a retroceder, de modo que el número no se imprima sobre la línea, depende de cuántos dígitos tenga el número. Este problema se resuelve fácilmente, porque el LOGO trata los números como si fueran palabras, de manera que podemos utilizar CONTAR para determinar la longitud del número, y luego emplearlo para determinar cuánto se ha de retroceder.

ESCRIBIR imprime un mensaje en la pantalla para gráficos. Toma tres entradas: las distancias de pasos x e y para cada carácter y el nombre a escribir. Se vale de una primitiva, STAMPCHAR, que imprime un carácter en la pantalla para gráficos, en la posición de la tortuga.

DIBUJAR.GRAFICO1 lleva a cabo la tarea de dibujar las barras. Toma cada elemento de uno en uno, selecciona el siguiente color a utilizar, somete a escala la altura y le pasa el verdadero trabajo a RELLENAR, que simplemente recorre la barra de arriba abajo rellenándola. ETIQUETA emplea ESCRIBIR para escribir las etiquetas verticalmente hacia abajo de la pantalla (las muy largas se desplazarán y aparecerán en la parte de arriba de la pantalla).

Gráficos de tarta

Los gráficos de tarta son otra forma muy común de representación gráfica. Si desea escribir un programa para dibujar un gráfico de tarta, he aquí algunas pistas:

- Los datos se obtienen exactamente de la misma forma que en el caso del gráfico de barras.
- La sección de cálculos consiste en totalizar los

números y determinar a partir de este dato qué porción les corresponderá de los 360° que constituyen la "tarta".

• Dibujar y rellenar los trozos de la tarta es bastante sencillo, pero en el Commodore 64, al menos, usted tendrá algunos problemas debido a la forma en que los colores se invaden entre sí. Es una buena idea aplicar la modalidad de "color doble"; tan sólo utilice DOUBLECOLOR en lugar de DRAW en el procedimiento que dibuja el gráfico. Si aun así tuviera problemas, deje un "agujero" de 10 unidades en el centro del gráfico de tarta.

El etiquetado se puede realizar de la misma forma que para el gráfico de barras, si bien podrían plantearse problemas al posicionar la tortuga antes de efectuar la escritura.

Después de que logre este gráfico, ¿por qué no tratar de escribir un programa que dibuje tanto un gráfico de barras como uno de tarta para los mismos datos, uno junto al otro?

Complementos al LOGO

Muchas versiones de LOGO no poseen un equivalente para STAMPCHAR, lo que hace que imprimir caracteres en la pantalla para gráficos resulte muy difícil.

Para las diferentes máquinas será necesario alterar los números de color y de tamaño del gráfico de barras. Todos estos detalles están reunidos en INIC, de modo que pueden tratarse todos juntos de una sola vez.

Para todas las versiones LCSI:
Utilice TYPE por PRINT1
Utilice EMPTYP por EMPTY?
Tal vez necesite emplear EQUALP en lugar del signo de igualdad (=)
SETXY debe ir seguida de una lista
IF posee una sintaxis diferente; por ej.:
IF EMPTYP :LISTADATOS [STOP]

Gráficos

Tanto los gráficos de barras como los gráficos de tarta constituyen unas formas muy útiles para visualizar información, si bien el LOGO no es ideal para estos fines debido a su poca velocidad

ENTRADA DE DATOS

	Londres	Nueva York
ENE	53	94
FEB	40	97
MAR	37	91
ABR	38	81
MAY	46	81
JUN	46	84
JUL	56	107
AGO	59	109
SEP	50	86
OCT	57	89
NOV	64	76
DIC	48	91

Datos de precipitaciones (en milímetros)

GRÁFICO DE BARRAS: Londres

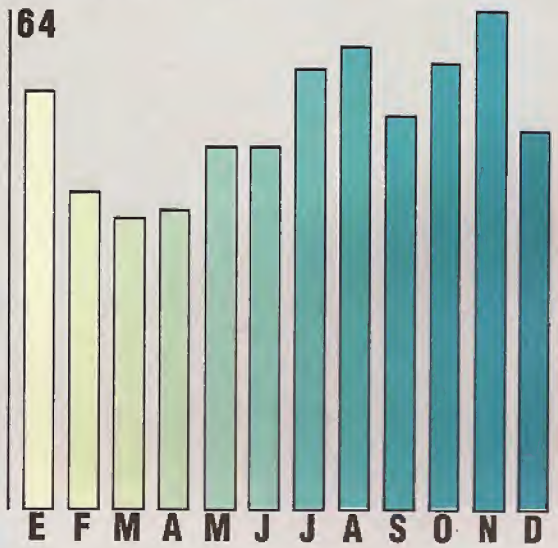


GRÁFICO DE TARTA: Londres

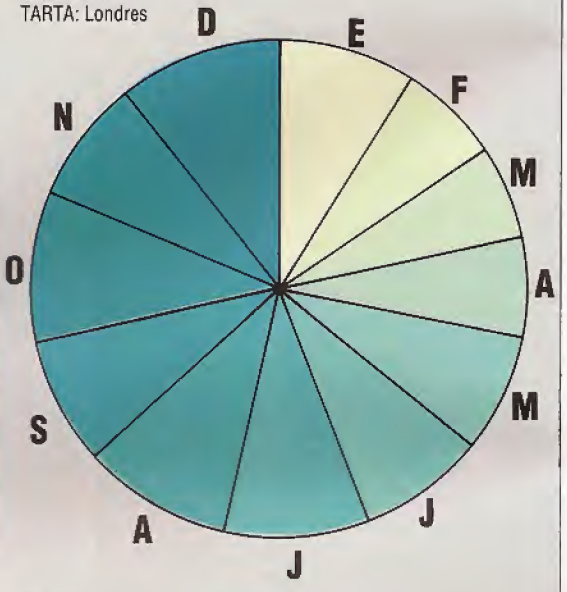


GRÁFICO DE BARRAS: Nueva York

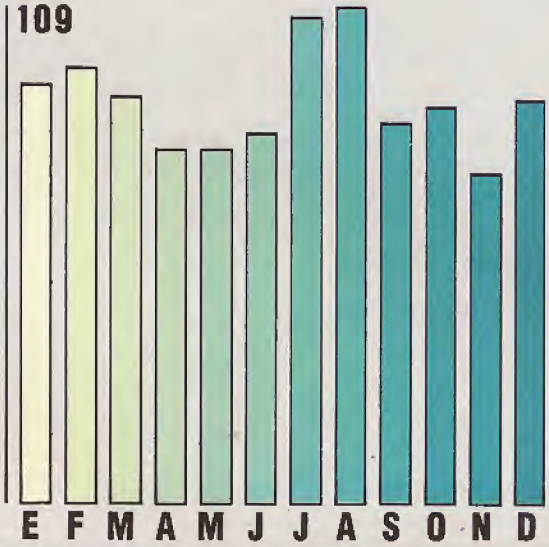
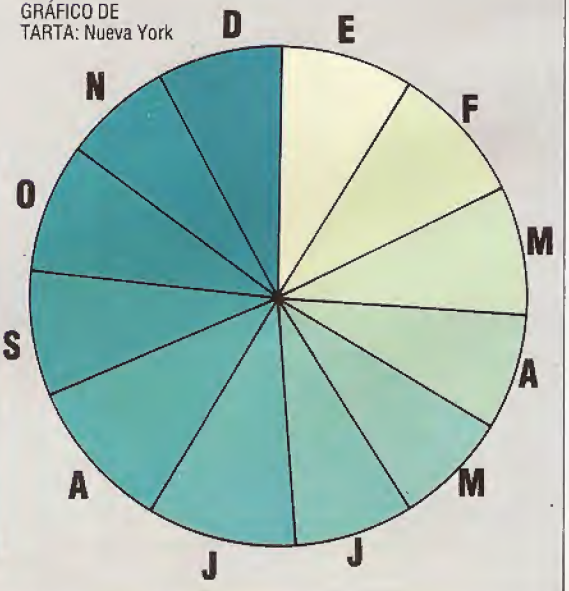


GRÁFICO DE TARTA: Nueva York



Programa para el gráfico de barras

```
TO GRAFICO.BARRAS
  INIC NODRAW
  TOMAR.ENTRADA CALCULAR
  DIBUJAR.GRAFICO
END

TO INIC
  MAKE "COLORES [0 1 2 5 6 0 1 2 5 6 0 1 2 5 6]
  MAKE "FONDO 11
  MAKE "DATOS []
  MAKE "ALTURA 160
  MAKE "ANCHURA 190
  MAKE "XORIGEN (-104)
  MAKE "YORIGEN (-40)
END

TO TOMAR.ENTRADA
  TOMAR.TITULO
  IF FIRST :TITULO = "FIN THEN STOP
  TOMAR.CANTIDAD
  MAKE "DATOS LPUT SENTENCE :TITULO
    :CANTIDAD :DATOS
  TOMAR.ENTRADA
END

TO TOMAR.TITULO
  PRINT "
  PRINT [(ENTRE FIN PARA TERMINAR)]
  (PRINT1 "TITULO: " ")
  MAKE "TITULO REQUEST
  IF EMPTY? :TITULO THEN TOMAR.TITULO
END

TO TOMAR.CANTIDAD
  (PRINT1 "CANTIDAD:"" ")
  MAKE "CANTIDAD REQUEST
  IF EMPTY? :CANTIDAD THEN TOMAR.CANTIDAD
    ELSE IF NOT NUMBER?
  FIRST :CANTIDAD THEN TOMAR.CANTIDAD
END

TO CALCULAR
  MAKE "MAX 0
  TOMAR.MAX :DATOS
  MAKE "ESCALA :ALTURA/:MAX
END

TO TOMAR.MAX :LISTADATOS
  IF EMPTY? :LISTADATOS THEN STOP
  IF LAST FIRST :LISTADATOS > :MAX THEN
    MAKE "MAX LAST FIRST :LISTADATOS
  TOMAR.MAX BUTFIRST :LISTADATOS
END

TO DIBUJAR.GRAFICO
  DRAW BACKGROUND :FONDO
  HIDETURTLE FULLSCREEN
  MAKE "ANCHURA ((ROUND ((:ANCHURA/
    COUNT :DATOS)/8))*8)-8
  PENUP SETXY :XORIGEN :YORIGEN
  DIBUJAR.EJE
  DIBUJAR.GRAFICO1 :COLORES :DATOS
  ETIQUETA :DATOS
END

TO DIBUJAR.EJE
  PENDOWN
  SETY YCOR+:ALTURA+10
  SETY YCOR-10
  SETX XCOR+4
  SETX XCOR-8
  PENUP
  SETX XCOR-(8*COUNT :MAX)
  ESCRIBIR 8 0:MAX
  SETX XCOR+4
  SETY :ORIGEN
END

TO ESCRIBIR :XINC :YINC :NOMBRE
  IF EMPTY? :NOMBRE THEN STOP
  STAMPCHAR FIRST :NOMBRE
  SETXY XCOR+:XINC YCOR+:YINC
  ESCRIBIR :XINC :YINC BUTFIRST :NOMBRE
END

TO DIBUJAR.GRAFICO1 :COLORESLAPIZ
  :LISTADATOS
  IF EMPTY? :LISTADATOS THEN STOP
  SETX XCOR+8
  PENCOLOR FIRST :COLORESLAPIZ
  PENDOWN
  RELLENAR ((LAST FIRST :LISTADATOS)*
    :ESCALA) :ANCHURA
  PENUP
  DIBUJAR.GRAFICO1 BUTFIRST
    :COLORESLAPIZ BUTFIRST :LISTADATOS
END

TO RELLENAR :LARGO :ANCHO
  IF :ANCHO=0 THEN STOP
  FORWARD :LARGO RIGHT 90
  FORWARD 1 LEFT 90
  BACK :LARGO RIGHT 90
  FORWARD 1 LEFT 90
  RELLENAR :LARGO :ANCHO-2
END

TO ETIQUETA :LISTADATOS
  ETIQUETA1 (:XORIGEN+8+ :ANCHURA/2)
    (:YORIGEN-10)
  :LISTADATOS
END

TO ETIQUETA1 :X :Y :LISTADATOS
  IF EMPTY? :LISTADATOS THEN STOP
  SETXY :X :Y
  ESCRIBIR 0(-10) FIRST FIRST :LISTADATOS
  ETIQUETA1 ( :X+ :ANCHURA+8)
    :Y BUTFIRST :LISTADATOS
END
```

Cuatro sensores

Ahora montaremos los sensores en el robot y escribiremos un programa

El control bidireccional de los motores paso a paso exige cuatro de las ocho líneas de datos de la puerta para el usuario de que disponemos. Ello nos deja cuatro líneas que se pueden utilizar para llevar información desde los sensores hasta el ordenador. Para conferirle más flexibilidad de operación a nuestro robot aplicaremos un "sistema de parches" para permitir la conexión de diferentes permutaciones de los sensores a las cuatro líneas de entrada disponibles. De momento, conectaremos cuatro sensores de microinterruptor y luego instalaremos dos sensores luminosos. Para seleccionar cualquier combinación de estos sensores (p. ej., dos sensores de microinterruptor y dos sensores luminosos), cablearemos cada sensor a un conector en la tapa del robot. Asimismo, se conectarán cuatro conectores a las líneas de datos (de D4 a D7) del enchufe D. Por consiguiente, podemos conectar el sensor apropiado para cualquiera de las cuatro líneas de datos mediante el empleo de un trozo de cable que se enchufe por un extremo en el conector del sensor y, por el otro, en uno de los conectores de las líneas de datos.

Para comprobar la construcción y el cableado de los cuatro microinterruptores podemos escribir un programa muy simple que explore los cuatro bits superiores del registro de datos y visualice los valores decimales de los bits enviados a 0. Ejecute el programa con los cuatro sensores conectados, vía el sistema de conectores con parches, a las líneas de datos de D4 a D7. El cierre de cualquiera de los microinterruptores provocará un cambio en la visualización en pantalla.

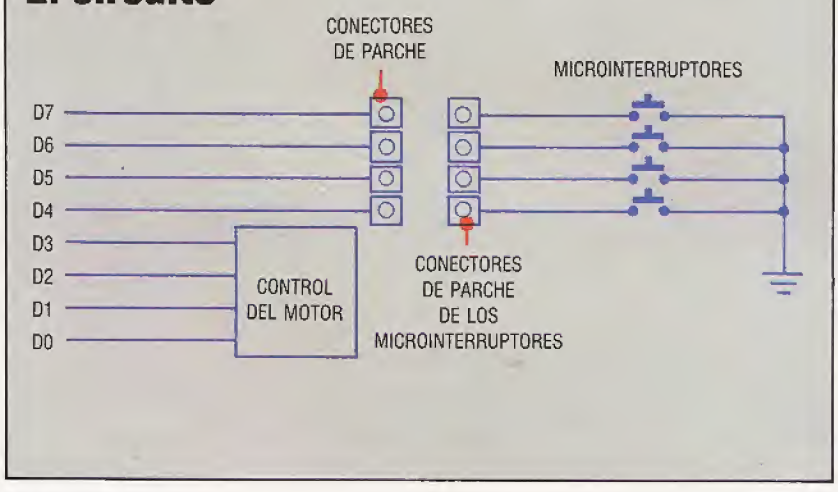
INSTALACIÓN

El tipo de microinterruptores que se menciona en la lista de componentes se puede adquirir en la mayoría de tiendas de componentes. En primer lugar, efectúe los cortes necesarios en la tapa de la carcasa del robot. Las ocho ranuras se deben cortar de modo que apenas quepan a través de ellas los conectores de abajo de cada microinterruptor. Los 10 agujeros de 5 mm de diámetro son para los conectores de parche, los cuales se pueden instalar en esta etapa, montando los cuatro conectores rojos en línea lo más cerca posible del enchufe D. Para este proyecto los microinterruptores se han de adaptar ligeramente. La palanca larga del microinterruptor debe inclinarse cuidadosamente, utilizando un par de pinzas, de modo que se forme un ángulo recto. Debe tenerse cuidado en asegurar que la inclinación no quede demasiado cerca de la caja del microinterruptor, porque en ese caso la palanca no cerrará el microinterruptor una vez que éste esté montado en la tapa. De la parte trasera de éste sobresalen dos conectores. De éstos, el de arriba es el conector NC ("normalmente cerrado"). Dado que en nuestro proyecto no lo vamos a utilizar, quizá sea mejor quitarlo, ya sea quebrándolo o cortándolo con una pequeña sierra para metales. El conector de abajo, el NO ("normalmente abierto") sí se utiliza y se lo debe inclinar en ángulo recto junto a la carcasa del interruptor. Ello asegurará que pase, junto con el COM ("conector común"), a través de las ranuras cortadas a tal fin en la tapa de la carcasa del robot. Una vez efectuadas estas modificaciones en cada uno de los cuatro microinterruptores, éstos ya se pueden montar, tal como se indica, en cada esquina de la tapa. Los interruptores deben montarse de modo que las palancas activadoras se cuelguen sobre la parte delantera y trasera de la carcasa del robot, y deben ser encolados en su sitio utilizando un adhesivo adecuado ("supercola" o Cyanoacrylate)

```
10 REM **** PRUEBA SENSORES BBC ****
20 MODE 7:OP=-1:RDD=&FE62:REGDAT=&FE60:?RDD=15
30 PE=240-(?REGDAT AND 240):IF PE=OP THEN 30
40 CLS:PRINT PE:OP=PE:GOTO 30
```

```
10 REM **** PRUEBA SENSORES CBM ****
20 OP=-1:RDD=56579:REGDAT=56577:POKE RDD,15
30 PE=240-(PEEK(REGDAT)AND 240):IF PE=OP THEN 30
40 PRINT CHR$(147):PRINT PE:OP=PE:GOTO 30
```

El circuito

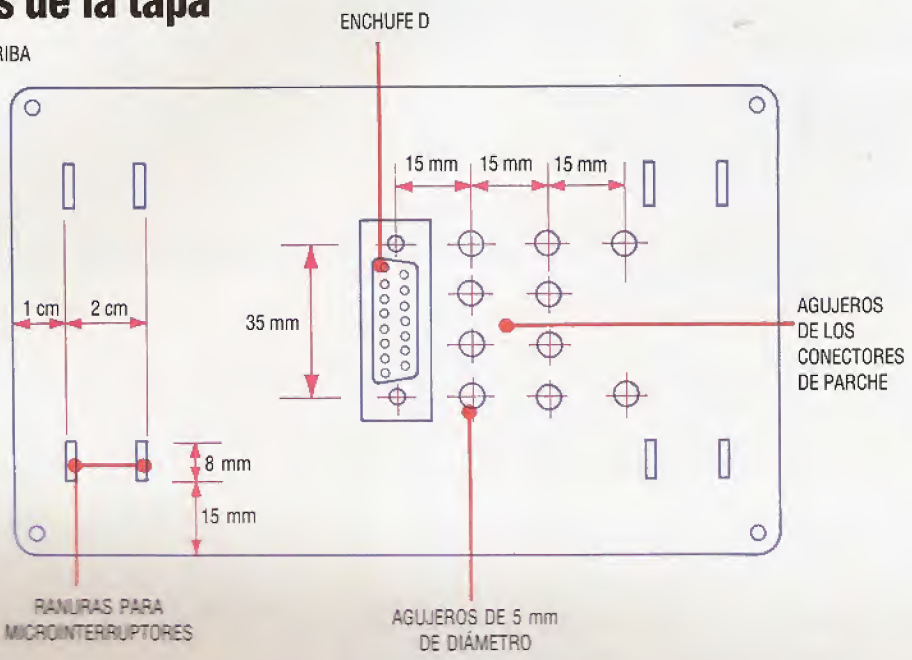


El circuito

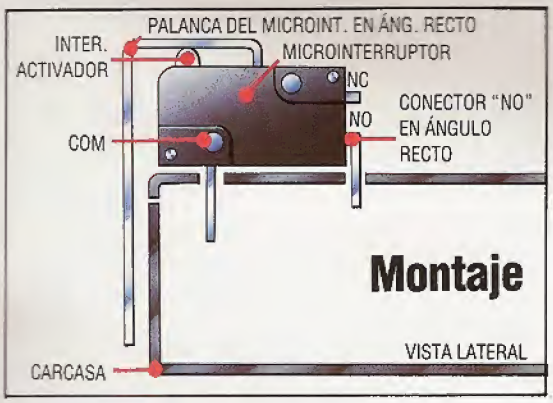
El circuito que conecta los sensores de microinterruptor con el sistema del robot es muy sencillo. Las líneas de datos de D4 a D7 están conectadas a los 4 conectores montados en la tapa del robot; las líneas de datos de D0 a D3 se utilizan para el control del motor. Uno de los lados de cada microinterruptor está conectado a un grupo similar de 4 conectores; el otro lado va conectado a una tierra común. De requerirse los 4 interruptores, se los puede parchear en las líneas de datos mediante 4 cables de parche. Si los 4 bits superiores del registro de datos de la puerta para el usuario se establecen en entrada, entonces se suelen mantener altos (en 1). El cierre de cualquier microinterruptor parcheado en el sistema conectará a tierra la línea de datos, poniendo bajo (en 0) el bit correspondiente

Agujeros de la tapa

VISTOS DESDE ARRIBA



Kevin Jones

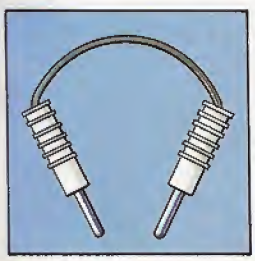


Montaje

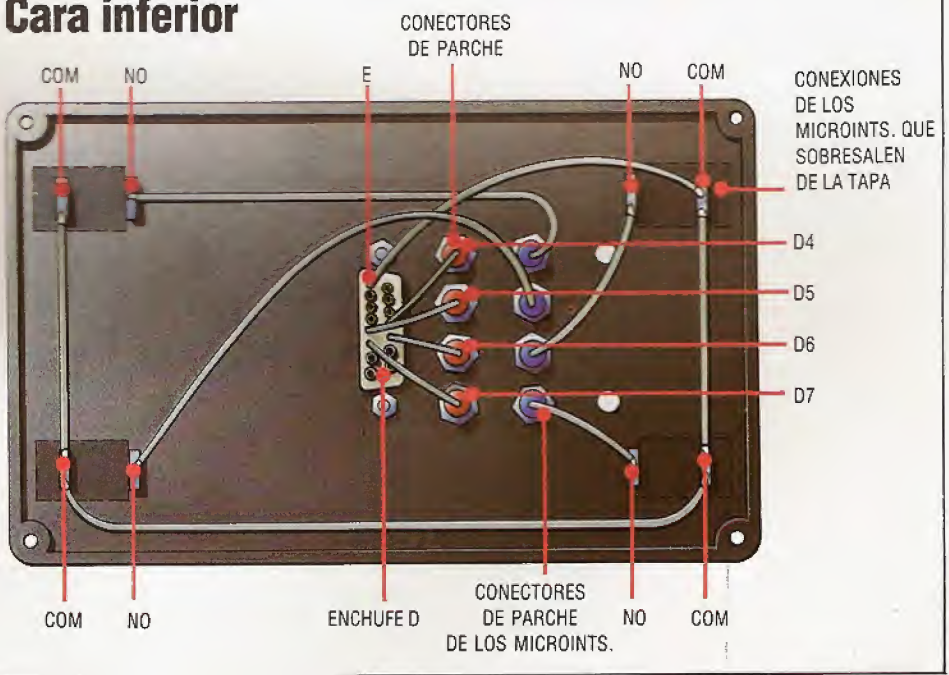
Lista de componentes

Cantidad	Artículo
8	enchufe 2 mm
4	conector rojo 2 mm
6	conector azul 2 mm
VARIOS	
4	microinterruptor
1 m	cable plano de 4 vías

Con los microinterruptores ya montados, dé la vuelta a la tapa. Suelde trocitos de cable revestido a cada uno de los 4 conectores rojos y conecte éstos a las patillas del enchufe D. Los 4 conectores COM deben estar unidos entre sí, y la patilla de tierra del enchufe D a un circuito circular. Luego conecte cada conector NO al conector azul adecuado. Construya 4 cables de parche para utilizar con el sistema



Cara inferior



Kevin Jones

Mapa de memoria del BBC

El presente capítulo está dedicado al empleo que el OS del BBC Micro hace de la memoria

El BBC Micro utiliza varias áreas de la memoria, muchas de ellas con varias funciones. Un ejemplo típico es la página &9 de la memoria (o sea, el área de la RAM que va de la posición &900 a la &9FF). Ésta sirve, según las veces, como buffer de salida del RS232, buffer de salida de la cassette, zona de trabajo para la voz y de zona de trabajo para el sonido ampliado. ¡Nadie puede acusar a Acorn de despilfarro de memoria!

Otro ejemplo es el bloque que va desde &0E00 a &1900. Esta área, cuando se emplea la cassette como sistema de archivo, queda libre para ser usada como espacio libre para programas BASIC. Pero si lo que se emplea es el disco, sirve de zona de trabajo del sistema de archivos en disco, reduciendo el total de memoria reservada a los programas BASIC en más de 2,5 K. Lo cual es una dificultad con los Modos del 0 al 2, pues ya queda previamente reducida la memoria.

Un último ejemplo del multiuso de la memoria está en el área entre &8000 y &BFFF. Si se acopla una ROM DFS, un procesador de textos o una ROM de utilidades en la máquina, la zona que ocupará será ésta. Pero también es aquí donde reside la ROM del intérprete de BASIC. Éstas se llaman ROM paginadas y sólo una de ellas puede estar activa a la vez. La ROM requerida es seleccionada por el sistema operativo, quien la "conecta" o "pagina". Normalmente la ROM del BASIC ya está paginada y el intérprete funcionando, lo que nos permite escribir y ejecutar programas en BASIC. Pero cuando se ejecuta la instrucción DFS, el sistema operativo pagina la ROM del DFS, ejecuta la instrucción y vuelve a paginar la ROM del BASIC. Esta última sencillamente se ignora durante la ejecución de la orden DFS. Otras ROM tales como las del TFS (Telesoftware Filing System), también para archivos, o del chip procesador de textos Wordwise, también ocupan el mismo espacio de memoria y son paginadas por el OS según se las requiera. Probablemente debido a que el BBC Micro emplea la misma área de memoria para múltiples fines, queda algo más de memoria para el usuario.

Memoria del usuario

Una vez que el OS ha tomado posesión de su parcela de memoria, el resto es la memoria disponible para el usuario. La variable BASIC llamada PAGE (página) contiene la dirección de inicio del área para programas BASIC, y la variable HIMEM (*high memory*: parte alta) señala el inicio de la memoria para visualización en pantalla; el espacio intermedio es el que queda para los programas BASIC, las variables y los programas en código máquina. Digite esta instrucción para conocer dichas direcciones y la memoria disponible para los programas BASIC:

```
PRINT~PAGE,~HIMEM~(HIMEM-PAGE)~"BYTES LIBRES"
```

En la tabla siguiente se proporciona una breve descripción de los distintos usos de las diversas áreas de almacenamiento para código máquina.

Qué áreas están disponibles para programas en código máquina	Tipo de sist. de archivo	
	Casset.	Disco
&0D00 – &0DFF Página &D: sólo cuando no existen ROM paginadas en la máquina	(✓)	X
&0B00 – &0CFF Pgs. &B y &C: cuando no se emplean en el programa caracteres definidos por el usuario o teclas de función del usuario	(✓)	(✓)
&0A00 – &0AFF Página &A: cuando no se está empleando una interface en serie	(✓)	(✓)
Espacio guardado en el área para programas BASIC por la instrucción DIM		
✓= Sí (✓) = Sí, con reservas X No empleado		

El almacenamiento del código máquina en el área para programas BASIC es la mejor solución para muchas rutinas empleadas junto con programas BASIC.

Una vez introducido en la memoria el código máquina, se necesita por lo general una zona de trabajo para poderlo ejecutar. Muchos programas en código máquina para el 6502 exigen la página cero como zona de trabajo, principalmente porque algunas instrucciones de direccionamiento indexado necesitarán posiciones de la página cero. Acorn ha empleado largamente esta página para el OS, pero algunas posiciones se han reservado para la programación en lenguaje máquina. No es de gran utilidad para el usuario una descripción byte por byte de lo que hace cada una de las posiciones del OS, y no se recomienda el acceso directo a ellas: las posiciones útiles deben accederse por medio de llamadas al OS, lo que significa una fuente de problemas a la hora de ejecutar un programa en código máquina escrito para una versión diferente del OS.

Entrada al OS

Las dos rutas principales que podemos tomar para el empleo de las rutinas contenidas en el OS del BBC son las rutinas OSBYTE y OSWORD:

• OSBYTE nos permite modificar el comportamiento de muchas rutinas del OS, pasando códigos de control y parámetros en los registros A, X e Y del 6502. En BASIC podemos acceder a las rutinas OSBYTE a través de la instrucción *FX. A la instrucción *FX siguen dos o tres números: el primer número es el control o código de función que se pasa al registro A; el segundo es el número que se pasa al registro X, y el tercero el que se pasa al registro Y. El tercer parámetro no es necesario en todas las llamadas de OSBYTE. En código máquina esta rutina es llamada en la dirección &FFF4. Estas dos versiones tienen funciones equivalentes:

BASIC	Assembly
*FX4,1	LDA # 4 LDX # 1 JSR &FFF4

El valor contenido en el registro A define exactamente lo que hará una llamada determinada a OSBYTE. La llamada ilustrada, por ejemplo, afecta a la acción de las teclas del cursor: el parámetro colocado en el registro X especifica el que las teclas retengan su función editora normal o el que den simplemente un código ASCII.

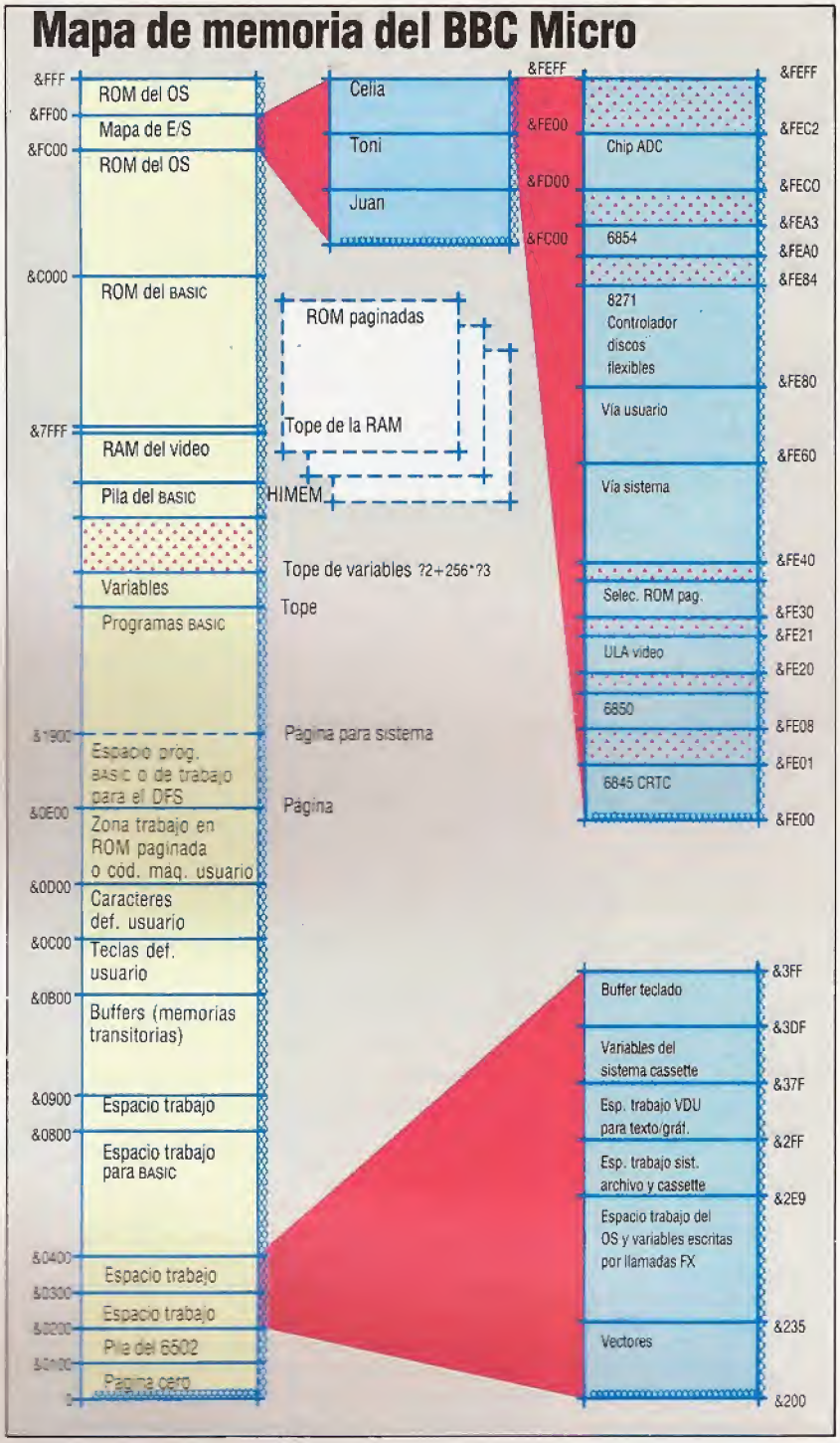
Desgraciadamente no todas las rutinas OS pueden ser sometidas al OSBYTE. La mayor desventaja está en que el número de parámetros que pueden pasarse a la rutina es limitado. Si se desconoce el contenido del registro A, que es el que indica al OS qué rutina deseamos emplear de entre las de OSBYTE, sólo podemos pasar dos parámetros a los registros X e Y. Para pasar algo más que esto nos serviremos de la segunda rutina, la OSWORD.

• La OSWORD permite cosas tales como generar sonidos, leer y escribir sobre disco, etc. Aquí reside la diferencia entre la OSBYTE y la OSWORD. Aquélla se ocupa de la *manera* en que el OS realiza ciertas tareas, y ésta nos permite la realización concreta de tales tareas. OSWORD obtiene sus parámetros de un *bloque de parámetros* al que apuntan los registros X e Y del 6502 al ceder control a la rutina OSWORD. Este bloque de parámetros se sitúa en la RAM y su tamaño y organización dependen de la llamada OSWORD que se haga. El registro A contiene un código de función que determina cuál de entre las funciones de OSWORD ha de ser ejecutada por el OS. Una vez preparados registros y bloque de parámetros, la llamada a OSWORD se realiza en la dirección &FFF1. Los principales instrumentos de entrada del OS son, sin duda, OSBYTE y OSWORD, y por su singular importancia habremos de examinarlas más adelante con mayor detalle.

Desde BASIC se puede acceder a otras rutinas del OS mediante un asterisco (*) seguido de una instrucción. Este símbolo hace que la instrucción que sigue evite el intérprete de BASIC y sea pasada a una rutina OS de nombre OSCLI (*Operating System Command Line Interpreter*). Esta rutina se encarga de interpretar las órdenes OS digitadas llamando a las adecuadas rutinas del OS. Con frecuencia tales instrucciones se denominan *instrucciones estrella* o *instrucciones* *. El cuadro adjunto al final del capítulo da una relación de tales instrucciones reconocidas por el BBC; las que no son reconocidas, sea porque no se escribieron correctamente o porque no se acompañaron del número adecuado de parámetros, suelen provocar un mensaje de error del tipo Bad Command (orden impropia).

Podemos pasar órdenes a la CLI (*Command Line Interpreter*: intérprete de línea de instrucción) por medio de la rutina OS o bien directamente. Los servicios de la OSCLI son dobles: primero, permite que pasemos las instrucciones del citado cuadro a la CLI vía código máquina si así lo deseamos, y, segundo, nos permite pasar variables alfanuméricas en BASIC a la CLI. Los siguientes programas ilustran ambos usos. Obsérvese que las variables numéricas enteras, A%, X% e Y% pasan sus valores directamente a los registros A, X e Y. De ellas, X% e Y% (o, lo que es lo mismo, los registros X e Y) apuntan a la

Mapa de memoria del BBC Micro



Liz Dixon

posición de memoria de la cadena de caracteres que se ha de tratar como una instrucción asterisco, y debe ser interpretada y ejecutada por la rutina OSCLI. Mientras X retiene el byte *lo* (inferior) de la dirección e Y retiene el *hi* o superior.

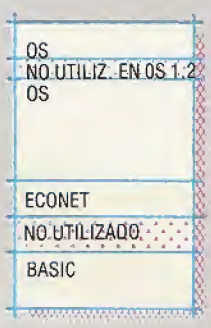
Versión BASIC	Versión BASIC+Assembly
10 DIM C 100 20 OSCLI=&FFF7 30 INPUT"ENTRAR INSTRUCCION",A$ 40 $C=A$ 50 X%=C MOD 256 60 Y%=C DIV 256 70 CALL OSCLI 80 GOTO 30	10 DIM C 100 20 OSCLI=&FFF7 30 FOR I%=0 TO 2 STEP 2 40 P%=&C00 50 [OPT I% 60 .code LDX #C MOD 256 70 LDY #C DIV 256 80 JSR OSCLI 90 RTS 100]:NEXT I% 110 INPUT "ENTRAR INSTRUCCION ", A$ 120 $C=A$ 130 CALL code 140 GOTO 110

La ejecución de este programa produce un aviso: el usuario digita una instrucción *, pulsa la tecla Return y la instrucción se ejecuta. La sentencia, más bien extraña, de la línea 10 de ambas versiones, DIM C 100, hace que el ordenador reserve 100 bytes de memoria, en el espacio reservado para las variables BASIC e inicializa la variable C con la dirección del inicio del bloque de memoria. Estos 100 bytes pueden ahora emplearse para almacenar programas en código máquina o, en un caso como éste, los datos de los programas en código máquina. La sentencia $C=A$ coloca los bytes que componen la cadena de la instrucción contenida en A$ dentro de ese bloque de 100 bytes, comenzando por el primer byte reservado por DIM. En ambas versiones los registros X e Y (o sea, las variables X% e Y%) son activados realizándose así la llamada a la OSCLI. La cadena de la instrucción se ejecuta seguidamente.

Este programa es la base de una rutina que se emplea en programas accionados por menú, donde puede resultar útil permitir al usuario obtener un catálogo de discos o cintas, por ejemplo, sin tener que abandonar el programa. La instrucción requerida se coloca sencillamente en las variables alfanuméricas y se pasa a la OSCLI para ser ejecutada. Si digitamos *A$ no funcionará. El OS intentará ejecutar una instrucción llamada *A$ que no existe.

Por medio de esta técnica es también factible pasar variables numéricas a una instrucción * mediante la función STR$ para convertirlas en alfanuméricas. Normalmente la CLI no aceptará ningún nombre de variable que se le pase; produce el mensaje de error Bad Command.

Toda instrucción * que no es reconocida por el OS es pasada a cualquier ROM que esté paginada. A cada una se le pregunta si la reconoce; si es así, la ejecuta. Las instrucciones que no se pasan de esta manera para ser ejecutadas son tratadas como órdenes incorrectas *sólo si* no se emplea un sistema rápido de archivo, como, por ejemplo, la unidad de disco. Si se emplea, el disco será inspeccionado para ver si contiene un archivo con el mismo nombre que el de la instrucción (excluido el *). Si lo contiene, el archivo se carga en la máquina y es tratado como un programa en código máquina. Esto puede ocasionar numerosos problemas en caso de que tal archivo no sea en realidad un programa. El ordenador suele "insistir" en estas situaciones hasta que usted no llega a sacarlo de su atasco. Si tal archivo no existe, entonces visualiza un mensaje de error Bad Command.



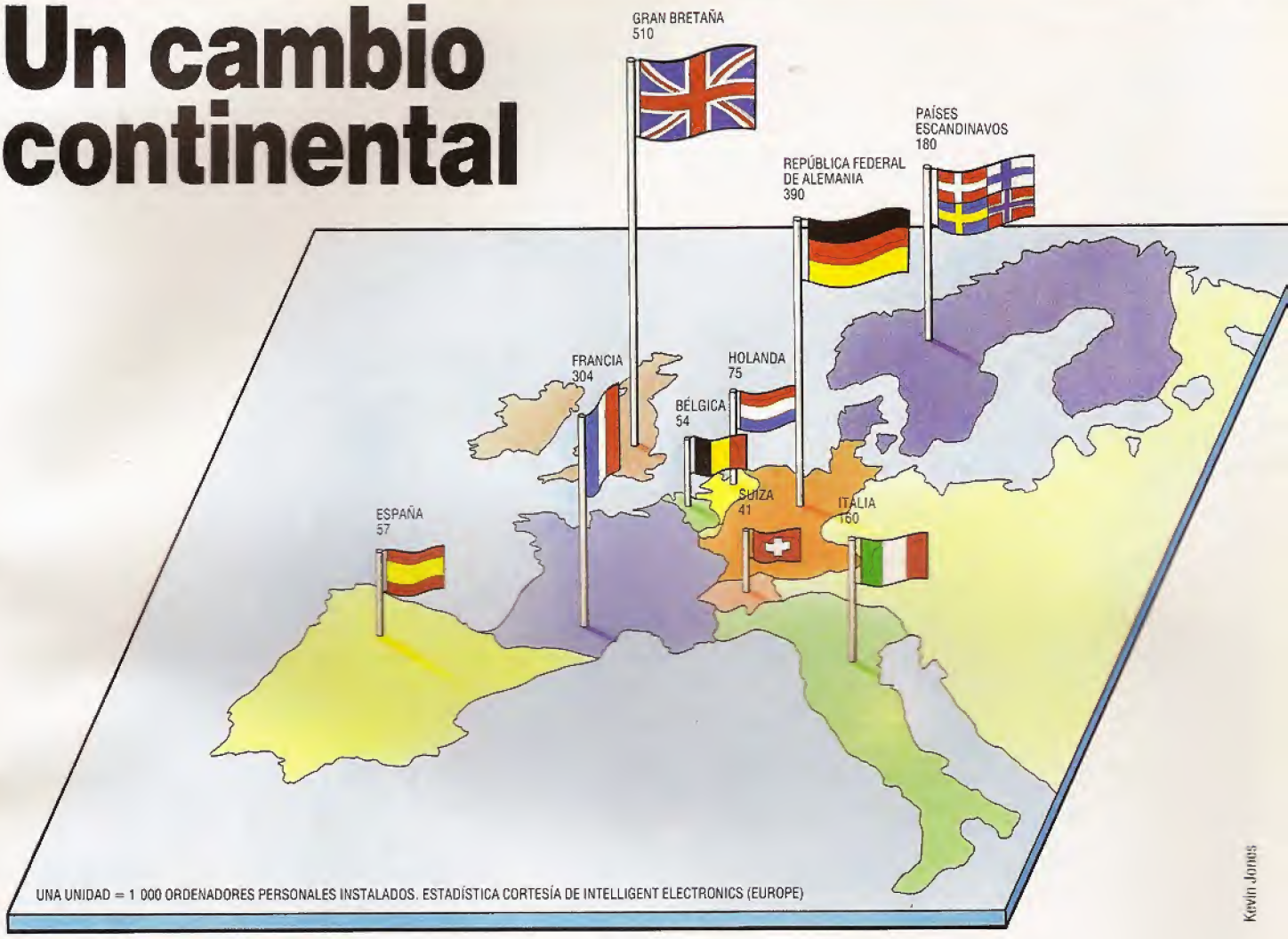
Uso de la página cero en el BBC Micro
Las limitadas facilidades de indexación del 6502 se complementan con su uso de la página cero (&0000 hasta &00FF) como una matriz de registros índices auxiliares. Esto conduce naturalmente a una densa utilización de estas posiciones por el OS, pero existen unos pocos bytes que no se emplean y se dejan para uso del programador

Las instrucciones*

Instrucción	Descripción y comentarios
*HELP	Da el número de la versión del BASIC. Puede también servir para obtener información sobre otras ROM paginadas que se han incorporado: así, *HELP DFS
*BASIC	Da entrada al lenguaje BASIC. Una variante de esta instrucción es, por ejemplo, *WORD, que da entrada a la ROM paginada View, en este caso
*CODE *LINE	Sólo en versión 1.2 del OS. Permite añadir nuevas órdenes
*KEY	Para programar teclas función rojas
*MOTOR n	Para controlar el relé del motor de la cassette: n=0 desconecta el relé y n=1 lo conecta
*ROM, *TAPE, *DISK, *NET	Para inicializar el adecuado sist. de archivo: *TAPE dará entrada al sist. de archivo en cass. de 1 200 baudios, y *DISK al de archivo en disco
*FX	Permite que el programador controle los valores de varias variables del OS y, gracias a eso, el comportamiento del OS
*RUN, *OPT *LOAD, *. *CAT, *SAVE	Instrucciones del sistema de archivo que examinaremos más tarde en otro capítulo
*SPOOL *EXEC	*SPOOL envía la salida de pantalla a la pantalla y a un archivo. *EXEC lee datos de un archivo como si se leyeran del teclado
*TV x,y	Afecta a la posición vertical de la pantalla y la interface de pantalla: x=0 no produce cambios en la posición vertical; x=1 mueve la pantalla hacia abajo una línea; x=255 la mueve hacia arriba una línea; y=0 activa el "interlace"; y=1 lo desactiva. Sus efectos se hacen operativos en el sig. cambio de modo y no cesan hasta que no se oprima CTRL-BREAK u otra instr. *TV

Una más detallada explicación de estas instrucciones las obtendrá el lector en la misma guía del usuario del BBC Micro

Un cambio continental



Kevin Jones

En esta ocasión analizaremos los pros y los contras que implica la traducción de programas del inglés a otras lenguas

Hasta hace poco tiempo casi todos los programas de software estaban escritos en inglés. El inglés era el idioma estándar de la informática debido al predominio de Estados Unidos en este campo durante la década de los cincuenta y los sesenta.

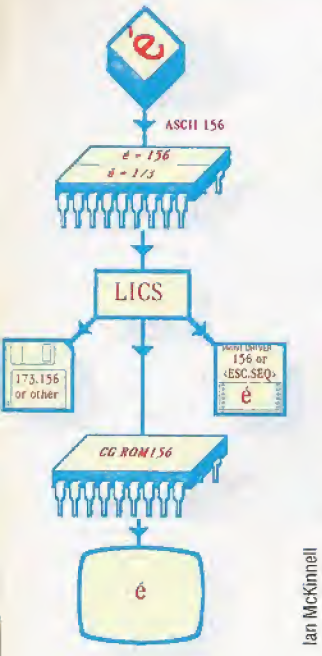
Sin embargo, la introducción del microordenador, a finales de los años setenta y principios de los ochenta, provocó grandes problemas en los países de habla no inglesa. Ello representó un gran obstáculo para la difusión de la microinformática a través de la Europa continental y creó un círculo vicioso. Las empresas no invertían en software que no comprendían ni podían utilizar con facilidad, y las firmas de software no gastaban el dinero necesario para traducir sus programas al idioma del país cuando las ventas previsibles no eran suficientes para cubrir los costos de desarrollo. El resultado fue que Gran Bretaña, disfrutando de la enorme ventaja de compartir un idioma común con Estados Unidos, se convirtió en el mercado más desarrollado de toda Europa para los microordenadores.

En la actualidad la situación está empezando a cambiar. Las firmas de software han comprendido el enorme mercado potencial de Europa continental y han empezado a traducir sus programas a los idiomas de los países en los cuales esperan vender sus productos. Lotus Software, como una de las mayores proveedoras de programas de gestión para IBM, fue una de las primeras empresas que produjo traducciones a otras lenguas.

El *Lotus 1-2-3* y el *Symphony* figuran entre los paquetes más vendidos de la nueva generación de "software integrado". Estos paquetes por lo general incorporan una hoja electrónica, base de datos y algunas capacidades para gráficos y tratamiento de textos. Los datos se pueden pasar entre cada una de estas aplicaciones, permitiendo, por ejemplo, almacenar información en una base de datos para tratarla en una hoja electrónica y poder luego incorporarla a un documento.

La traducción de uno de tales programas del inglés a, por ejemplo, el italiano, podría parecer relativamente sencilla: todas las instrucciones que aparezcan en la pantalla se han de traducir al idioma apropiado. Sin embargo, de inmediato surgen diversas dificultades. En primer lugar, si el texto está incorporado en el propio código fuente, encontrar el texto en 120 K de código en el que tanto el texto

Ordenadores personales en Europa
Debido a que Gran Bretaña partió con la ventaja de compartir una lengua común con Estados Unidos, hasta el momento las empresas británicas han superado ampliamente a las de Europa continental desde el punto de vista de la instalación de ordenadores personales. Parece ser, sin embargo, que el notable crecimiento de las ventas en Gran Bretaña está terminando justo cuando se produce el despegue del mercado en los otros países europeos: se calcula que para 1988 la República Federal de Alemania habrá superado a Gran Bretaña en la cantidad total de máquinas instaladas



Ian McKinnell

ASCII internacional
Para tratar con los juegos de caracteres específicos que emplean los distintos idiomas, Lotus ha desarrollado el LICS (Lotus International Character Set: juego internacional de caracteres Lotus), en el cual hay un código para cada carácter no perteneciente al idioma inglés. En un teclado francés, por ejemplo, la pulsación de la letra *é* (como en *café*) generaría un código 156; éste corresponde a 173 en LICS. Cuando este carácter se ha de enviar a la pantalla, el LICSA lo vuelve a decodificar a 156; la impresora podría comprender el código 156, o bien necesitar que se le enviara una secuencia de control tal como <e>, <ESC>, <BSPACE>, <'>. El texto se puede guardar en códigos ASCII ingleses, en LICS o en códigos ASCII de otro país

como el programa están representados por números constituye una tarea muy complicada. En segundo lugar, si el texto traducido es más extenso que el texto original en inglés (lo que sucede casi siempre) el programa necesitará más bytes para almacenarlo. Esto alterará las direcciones de todo el código subsiguiente, convirtiendo, por lo tanto, en un sin sentido los bucles y las llamadas a subrutinas.

Otro problema es la sintaxis. Cuando un usuario inglés desea operar sobre un archivo, la sintaxis es COMMAND (instrucción) seguido de FILENAME (nombre del archivo). Sin embargo, este enfoque no es el normal en otros idiomas. En alemán, por ejemplo, lo lógico es entrar primero el nombre del archivo, seguido por la instrucción. Se plantea un problema similar en la forma de entrar las fechas, lo que ha sido fuente de problemas incluso en Gran Bretaña. Tanto el *1-2-3* como el *Symphony* permiten el empleo de fechas en fórmulas para calcular cambios en los valores a través del tiempo. En Estados Unidos, el método normal para entrar los datos es mes/día/año. Sin embargo, en Gran Bretaña y en muchas otras partes de Europa el formato estándar para fechas es día/mes/año. A menos que el paquete de software se pueda manipular de modo que tenga en cuenta las diferencias en cuanto a la forma de entrar instrucciones y datos, el resultado será, en el mejor de los casos, confuso o, en el peor, un sin sentido total.

Los editores de software deben asimismo tener en cuenta los diferentes juegos de caracteres. El alfabeto francés incluye letras tales como *é* y *à*, mientras que el alemán y las lenguas escandinavas incluyen la letra *ä*. Para complicar aún más las cosas, los diferentes alfabetos colocan estas letras en otro orden, haciendo estragos en cualquier rutina incapaz de resolver estas diferencias.

Diseño de programa

Al traducir el *Symphony* a las principales lenguas europeas, Lotus decidió que la única forma razonable de abordar el problema era diseñar el programa de modo que permitiera una fácil traducción. Este enfoque no se adoptó con el paquete *1-2-3* anterior y, a consecuencia de ello, la empresa tuvo grandes dificultades para traducirlo. No obstante, el *Symphony* se ha traducido con total éxito al francés, al alemán y a las lenguas escandinavas y la empresa está trabajando ahora en una versión italiana.

Para superar los problemas de localización del texto en el código y tratar de comprimir las nuevas palabras en el espacio disponible, Lotus ha adoptado una construcción modular del programa. Dentro de éste hay dos divisiones: el código fuente que contiene las rutinas del programa y un segmento de datos que incluye el área de textos. Este sistema de aislar el texto del código fuente se conoce como *localización*. La organización del programa de acuerdo a este formato resuelve dos de las dificultades principales. Primero, el hecho de tener el texto en un segmento separado significa que se puede reservar espacio extra para cualquier diferencia en cuanto a extensión del texto, y éste puede ser extraído del programa con mucha más facilidad. Una utilidad extrae del código las áreas de texto; éstas se pueden entonces traducir y volver a incluir en el segmento de datos. Como ventaja adicional, el texto se puede reacomodar dentro de la sección de datos para dar cabida a las diferencias de sintaxis que requiera cada idioma.

Al traducir el texto propiamente dicho hay que considerar todavía otro punto. El *Symphony* permite entrar las instrucciones simplemente pulsando el primer carácter del nombre de la instrucción. Por lo tanto, cada instrucción debe empezar con una letra diferente. Además, las limitaciones de espacio significan que en un paquete en el cual la traducción literal sea demasiado larga, quizá sea necesario hacer alguna concesión.

También pueden surgir problemas cuando se traduce entre juegos nacionales de caracteres. Ya hemos visto que los diversos países poseen letras diferentes en sus alfabetos. Lo que agrava el problema es que para los códigos no existe ninguna estandarización acordada con carácter internacional. En los días de la cinta de papel, cuando eran comunes los códigos de siete bits, el estándar ASCII se utilizaba en todo el mundo casi sin ninguna excepción. Con el advenimiento del microordenador y de los códigos de ocho bits, la estandarización desapareció debido a que cada fabricante produjo su propia versión del "estándar" ASCII.

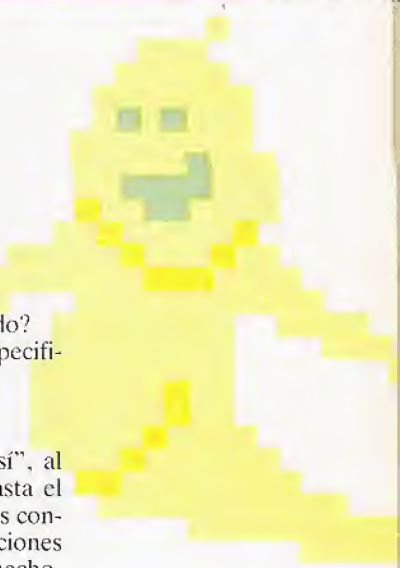
Esta práctica la han adoptado distintos países. Al adaptar el teclado del ordenador a su propio juego de caracteres, muchas naciones han sustituido algunos caracteres ingleses por sus propias letras. Por consiguiente, la comunicación entre ordenadores configurados para idiomas diferentes se está convirtiendo en un enorme problema. El código ASCII en alemán podría significar algo totalmente diferente en castellano.

Esta confusión la agravaron aún más los traductores del *Symphony*, ya que muchas de las instrucciones de una única pulsación de tecla utilizadas en el programa eran caracteres tales como @, que no existe en los teclados no ingleses. La solución que encontró para este problema la propia IBM en su PC es mantener pulsada la tecla ALT y digitar el código ASCII decimal en el teclado numérico, ¡anulando, de este modo, la ventaja que reportaba el tener instrucciones para las cuales sólo se había de pulsar una tecla!

Lotus decidió que la única forma de abordar este problema era desarrollar su propio juego de códigos, que se denomina LICS (Lotus International Character Set: juego internacional de caracteres Lotus). Este juego, de 250 caracteres, contiene todas las letras utilizadas en la mayoría de las lenguas europeas y está almacenado en todas las copias del *Symphony*. La traducción supone configurar el programa de modo que el código que reciba un teclado de lengua no inglesa se traduzca a LICS, y el programa pueda, en consecuencia, comprenderlo. Para simplificar el proceso de imprimir caracteres que no aparezcan en el teclado, Lotus ha conseguidor reducir estos caracteres a una única pulsación de la tecla ALT y un número entre 0 y 9.

El proceso de traducción de un paquete de gestión es una operación costosa y que consume mucho tiempo. La traducción lleva un promedio de nueve meses y es extraordinariamente cara. No obstante, las firmas de software ya no pueden permitirse el lujo de ignorar un mercado de 300 millones de personas como es el de Europa continental. A pesar de la inversión que requiere la traducción de un paquete de software, los beneficios que ésta reporta bien valen el esfuerzo.

Desde el Más Allá



Nos corresponde abordar el resto de la rutina del túnel y diseñar una subrutina para que aparezcan fantasmas al azar en el "bosque encantado" de nuestro juego

En el capítulo anterior analizamos los escenarios especiales que poseen una entrada al túnel: en estos escenarios se le ofrece al jugador la oportunidad de penetrar en el túnel o de retroceder hasta el sendero que lo condujo a la entrada. Si el jugador elige entrar en el túnel, entonces se llama a una nueva subrutina, en la línea 4655. Veamos ahora la subrutina que maneja la opción en la cual el jugador penetra en el túnel. Esta subrutina está escrita de acuerdo a ciertas reglas que estableció el diseñador del juego. En primer lugar, el jugador sólo puede atravesar el túnel si lleva consigo el farol; además, debe encender el farol para alumbrar su camino.

Dado que el jugador ha de poder impartir instrucciones mientras permanezca en el interior del túnel, la subrutina debe empezar con una secuencia que acepte la entrada de una instrucción y la descomponga para su proceso. Podemos permitir que el jugador utilice algunas de las instrucciones de entrada normales, como RECOGER, DEJAR, LISTA o FIN, pero en este punto hemos de tener cuidado. Por cuanto concierne al puntero del escenario, P, el jugador se halla todavía en la boca del túnel y, por lo tanto, está en condiciones de AVANZAR en ciertas direcciones permitidas. Por consiguiente, mientras nos encontremos dentro del túnel, debemos suprimir las instrucciones AVANZAR.

Al retornar de la subrutina "instrucciones normales", de haberse generado una instrucción AVANZAR se establecerá la "bandera de movimiento" (MF) y el valor de P habrá cambiado. Este efecto se puede anular simplemente restaurando a P el valor que tenía antes de que se llamara a la subrutina "instrucciones normales".

Habiendo manipulado satisfactoriamente las instrucciones normales, podemos pasar a considerar las instrucciones especializadas necesarias para esta situación en particular. Se puede permitir que el jugador utilice una instrucción RETROCEDER para regresar hasta la entrada al túnel que traspasó al entrar. La única otra instrucción que permitiremos es ENCENDER, o una variación, USAR. Si la instrucción impartida no es ninguna de éstas, la rutina producirá un mensaje general NO COMPRENDO antes de desandar el bucle en espera de otra instrucción.

Si la instrucción es ENCENDER o USAR, entonces tendremos que efectuar varias comprobaciones antes de obedecer la instrucción:

1. ¿Es el objeto especificado un objeto válido?
2. ¿Lleva consigo el jugador el objeto especificado?
3. ¿Es el farol el objeto especificado?

Si la respuesta a todas estas preguntas es "sí", al jugador se le permitirá atravesar el túnel hasta el otro extremo, puesto que se han satisfecho las condiciones para atravesarlo. Estas comprobaciones del objeto ya nos resultan familiares. De hecho, son casi idénticas a las empleadas en las rutinas RECOGER y DEJAR. Por lo tanto, para llevar a cabo estas verificaciones podemos utilizar rutinas escritas anteriormente.

```
4700 REM ** ENTRAR AL TUNEL **
4705 SN$="ENTRAS EN EL TUNEL PERO ESTA DEMASIADO OSCURO
     COMO"
4710 SN$=SN$+"PARA ENCONTRAR EL CAMINO":GOSUB5500
4725 PRINT:INPUT"INSTRUCCIONES";IS$
4730 GOSUB2500:REM DESCOMPONER INSTRUCCION
4732 :
4735 IF F=0 THEN 4725:REM INSTRUCCION NO VALIDA
4740 OP=P:GOSUB3000:REM INSTRUCCIONES NORMALES
4745 IF MF=1 THEN SN$="ESTA TAN OSCURO QUE SOLO PUEDES
     VER":P=OP
4747 IF MF=1THENSN$=SN$+" LA ENTRADA DEL
     TUNEL":GOSUB5500:MF=0:GOTO 4725
4750 IF VF=1 THEN 4725:REM INSTRUCCION OBEDECIDA
4755 IF VB$="RETROCEDER" AND P=4 THEN MF=1:P=6:
     RETURN
4760 IF VB$="RETROCEDER" AND P=1 THEN MF=1:P=9:
     RETURN
4762 IF VB$<>"USAR" AND VB$<> "ENCENDER"THEN SN$="NO
     COMPRENDO"
4765 IF VB$<>"USAR" AND VB$<> "ENCENDER"THEN
     GOSUB5500:GOTO4725
4777 :
4780 REM ** BUSCAR FAROL **
4790 GOSUB5300:REM OBJETO VALIDO?
4795 OV=F:GOSUB5450:REM LLEVA EL OBJETO?
4797 IF F=0 THEN SN$="NO HAY NINGUN"+WS:GOSUB5500:
     GOTO4725
4800 IF HF=0 THEN SN$="TU NO TIENES EL "+IVS(F,1):GOSUB5500:
     GOTO4725
4810 REM ** EL OBJETO ES EL FAROL? **
4815 IF F<>2 THEN SN$="LA "+IVS(F,1)+" NO
     SIRVE":GOSUB5500:GOTO4725
4835 REM ** EXITO **
4840 SN$="USAS EL FAROL PARA ILUMINARTE EL CAMINO A TRAVES
     DEL TUNEL"
4845 SN$=SN$+" Y FINALMENTE SALES POR LA SALIDA.":
     GOSUB5500
4850 IF P=1 THEN MF=1:P=4:RETURN
4855 IF P=4 THEN MF=1:P=1:RETURN
```

Fenómenos sobrenaturales

Además de tener escenarios especiales, como las entradas al túnel, podemos programar peligros o acontecimientos al azar. Llegados a este punto, en el desarrollo de nuestro juego *El bosque encantado* no hemos mencionado fantasmas, ni los mismos aparecen en el mapa del mundo de aventuras para el juego. En cambio, los fantasmas se le aparecen al azar al jugador a medida que va recorriendo el bosque y sólo se los puede mantener a raya emprendiendo una acción extravagante. Antes de analizar detalladamente la rutina "de los fantasmas",

consideremos cómo podemos incorporar en la estructura del programa principal las rutinas para generar apariciones aleatorias. El bucle principal del programa llama a una subrutina de la línea 2700 para constatar si un nuevo escenario es especial en algún sentido o no. Éste es también el mejor lugar para incorporar el siguiente trozo de código, que tiene como finalidad decidir si el programa debe generar fantasmas al azar:

```
2707 REM ** FANTASMA AL AZAR **
2710 IF P >4 AND RND(1)<0.1 THEN GOSUB 4290:RETURN
```

La línea 2710 asegura, en primer lugar, que el escenario actual no haya sido ya designado como especial, dado que si los fantasmas aparecieran en la mitad de rutinas especiales harían que la vida resultara muy complicada. Si el escenario es común, utilizando la instrucción RND existe una posibilidad entre 10 de que el programa produzca un fantasma. Las instrucciones RND generan números *seudoaleatorios*, así llamados porque el patrón de números generados desde la conexión a la red del ordenador es predecible. Para hacer que la secuencia sea menos predecible, utilizamos la instrucción RND con un operando negativo en el caso del BBC Micro y del Commodore 64, y la instrucción RANDOMISE para el Spectrum (véase "Complementos al BASIC" en la página contigua).

```
207 R=RND(-1)
```

Si se llama a la rutina de los fantasmas, entonces entramos en otro escenario especial en el cual el jugador se enfrenta a la aparición fantasmal. La rutina sigue el procedimiento usual: genera un mensaje inicial, solicita una instrucción y la descompone en el verbo y el resto de la frase. Las instrucciones normales son tratadas por la subrutina estándar; pero nuevamente se suprime la instrucción AVANZAR: un mensaje informa al jugador que, al haberse quedado paralizado por el terror, no se puede mover.

Red de seguridad

En esta etapa se pueden tratar nuevas instrucciones. Al igual que las otras rutinas de tratamiento de escenarios especiales, la calidad del juego terminado depende de cuánto esfuerzo de programación se dedique al diseño de estas rutinas. Cualquier instrucción que no sea directamente útil en la rutina se puede tratar mediante la red de seguridad NO COMPRENDO. No obstante, con un esfuerzo de programación adicional, podemos manejar instrucciones que cabría esperar del jugador pero que no serán de ayuda en su situación. En la rutina "fantasmas" se emplea un ejemplo de este enfoque.

Si un jugador se encuentra con un fantasma, su primera idea podría ser LUCHAR o MATAR (¡en el caso de que uno pudiera matar a los fantasmas!). La rutina "fantasmas" trata estas dos instrucciones llamando a una subrutina especial. Esta subrutina simplemente visualiza un mensaje que señala que estas instrucciones no le son de ayuda al jugador, pero lo hace de tal forma que es sustancialmente más atractiva que limitarse a expresar un escueto NO COMPRENDO.

```
4290 REM **** S/R FANTASMA AL AZAR ****
4295 SF=1:GC=0
4300 SN$="SIENTES QUE UN ESCALOFRIO TE RECORRE"
4305 SN$=SN$+" LA COLUMNA VERTEBRAL. DE PRONTO UNA
     APARICION BLANCA"
4310 SN$=SN$+" SALE DE ATRAS DE LOS ARBOLES Y"
4315 SN$=SN$+" AVANZA HACIA TI":GOSUB5500:REM
     FORMATO
4320 :
4325 SN$="EL FANTASMA SE ACERCA MAS Y MAS":GOSUB
     5500
4330 GC=GC+1:IF GC>4 THEN GOSUB4455:REM
4335 PRINT:INPUT"INSTRUCCIONES";IS$
4340 GOSUB2500:REM DESCOMPONER INSTRUCCION
4345 IF F=0 THEN 4325:REM SIGUIENTE INSTRUCCION
4350 OP=P:GOSUB3000:REM ANALIZAR INSTRUCCION
4355 IF MF=1 AND VB$="AVANZAR" THEN GOSUB4400: GOTO 4325
4357 IF MF=1 AND VB$="MIRAR" THEN GOSUB2000:GOSUB2300:
     GOTO4325
4360 IF VF=1 THEN 4325:REM SIGUIENTE INSTRUCCION
4365 REM ** PALABRAS DE INSTRUCCION NUEVAS **
4370 IF VB$="MATAR" OR VB$="LUCHAR" THEN GOSUB4425:GOTO
     4325
4375 :
4385 IF VB$="CANTAR" THEN GOSUB4500:RETURN
4390 SN$="NO COMPRENDO":GOSUB5500:GOTO4325
4395 :
4400 REM ** TRATAR DE MOVERSE **
4405 SN$="ESTAS PARALIZADO POR EL TERROR Y NO PUEDES"
4410 SN$=SN$+" MOVERTE...TODAVIA":MF=0:GOSUB5500: P=OP
4415 RETURN
4420 :
4425 REM ** LUCHAR O MATAR **
4430 SN$="EL FANTASMA ES UN SER SOBRENATURAL"
4435 SN$=SN$+" Y SE RIE DE TUS DEBILES INTENTOS"
4440 SN$=SN$+" POR HERIRLO":GOSUB5500
4445 RETURN
4450 :
4455 REM ** MUERTE **
4460 SN$="EL DOLOR QUE SIENTES EN EL PECHO SE VUELVE
     INSOPORTABLE"
4465 SN$=SN$+" Y TE DESPLOMAS SOBRE EL SUELO CUBIERTO DE
     HOJAS DEL BOSQUE.":GOSUB5500
4470 SN$="TU ESPIRITU SE DESPRENDE DE TU CUERPO INERTE"
4475 SN$=SN$+" Y TE ALEJAS FLOTANDO ENTRE LA NIEBLA PARA
     UNIRTE"
4480 SN$=SN$+" A LAS OTRAS ALMAS ATORMENTADAS DE"
4485 SN$=SN$+" EL BOSQUE ENCANTADO.":GOSUB5500
4490 END
```

Una pequeña trampa

De impartirse alguna de las instrucciones normales, o instrucciones que no le sirven de nada al jugador, la rutina las obedecerá, si le es posible, y saltará hacia atrás del bucle para una nueva instrucción. Esta rutina tiene una pequeña trampa, porque lleva la cuenta del número de instrucciones impartidas por el jugador mientras se mide con el fantasma. De impartir más de cuatro instrucciones, el fantasma mata al jugador. El único medio de que dispone el jugador para escapar es CANTAR una canción. Si el jugador opta por cantar, se le pide que elija entre tres canciones, una de las cuales (escogida al azar) apaciguará al fantasma. Sin embargo, si el jugador elige una canción errónea, su espíritu se unirá al ejército de almas atormentadas que se han extraviado en *El bosque encantado*:

```
4500 REM ** CANTAR **
4505 SN$="SABES TRES CANCIONES. CUAL ELEGIRIAS ?":
     GOSUB5500
4510 SN$="1) EL TEMA MUSICAL DE 'LOS
     CAZAFANTASMAS'":GOSUB5500
4515 SN$="2) 'HAY UN FANTASMA EN MI CASA'":GOSUB5500
4520 SN$="3) 'BAJANDO POR EL RIO SWANEE'":GOSUB5500
4525 PRINT:INPUT"ELIGE UNA";C$
4530 IF VAL(C$)>3 OR VAL(C$)<1 THEN PRINT:PRINT"NO
     VALE":GOTO4525
4535 CR=INT(RND(1)*3)+1
4537 IF CR<>VAL(C$)THEN GOSUB4542:REM CANCION EQUIVOCADA
4540 GOSUB4565:REM CORRECTA
4542 REM **** S/R CANCION EQUIVOCADA ****
4545 SN$="EL FANTASMA TIENE UNA ESPECIAL AVERSION POR "
4550 SN$=SN$+"ESA MELODIA Y SE ABALANZA SOBRE TI."
     GOSUB5500
4555 GOSUB 4455:REM MUERTE
4560 :
4565 REM ** CANCION CORRECTA **
4570 SN$="EL FANTASMA SE APACIGUA AL OIRTE CANTAR LA
     MELODIA"
4575 SN$=SN$+" Y SE EVAPORA EN EL AIRE":GOSUB5500
4580 RETURN
```

Fantasmas realizados por Liz Dixon

Listados Digitaya

```
2690 IF P=37 THEN2780:REM TABLA VECTORES
2700 IF P>7 THEN 2750:REM BICHO AZAR

2740 REM ** BICHO AL AZAR **
2750 RA=RND(TI)
2760 IF RA<.5 THEN GOSUB 5420:REM BICHO
2770 RETURN
2780 REM ** TABLA VECTORES **
2790 SF=1
2800 SN$="ERES LLEVADO A GRAN VELOCIDAD HASTA UN NUEVO
     ESCENARIO":GOSUB5680
2810 FOR J=1TO1000:NEXT:REM PAUSA
2820 P=INT(RND(TI)*40+7)
2830 MF=1:RETURN

4550 REM **** ALU ****
4560 SF=1
4570 RN=INT(RND(TI)*3+1)
4580 IF RN=1 THEN CD$="AND"
4590 IF RN=2 THEN CD$="OR"
4600 IF RN=3 THEN CD$="NOT"
4610 SN$="MONTADOS SOBRE LA PARED HAY TRES BOTONES
     ROTULADOS"
4620 SN$=SN$+"'AND', 'OR' Y 'NOT'. SE PUEDE GANAR ACCESO
     AL"
4630 SN$=SN$+"ACUMULADOR PULSANDO EL BOTON
     ADECUADO"
4640 GOSUB5880:REM FORMATO
4650 :
4660 REM ** INSTRUCCIONES **
4670 PRINT:INPUT"INSTRUCCIONES";IS$
4680 GOSUB1700:GOSUB1900:REM ANALIZAR
4690 IF MF=1 THEN RETURN:REM IRSE
4700 IF VF=1 THEN 4670:REM SIGUIENTE INSTRUCCION
4710 IF VB$="USAR"OR VB$="PULSAR"THEN4740
4720 PRINT" NO COMPRENDO":GOTO4670
4730 :
4740 REM ** INSTRUCCION VALIDA **
4750 IF VB$="PULSAR" THEN 4930
4760 REM ** LA INSTRUCCION ES 'USAR' **
4770 GOSUB5730:REM ES VALIDO EL OBJETO
4780 IF F=0THENPRINT"NO HAY NINGUN ";NN$:GOTO4670:REM
     SIGUIENTE INSTRUCCION
4790 :
4800 REM ** ES EL OBJETO EL LIBRO DE CODIGOS **
4810 IF F=7 THEN4850:REM OK
4820 SN$="TU "+IV$(F,1)+"NO SIRVE DE NADA":GOSUB5880
4830 GOTO4670:REM SIGUIENTE INSTRUCCION
4840 :
4850 OV=7:GOSUB5830:REM LLEVA EL LIBRO DE CODIGOS
     CONSIGO
4860 IF HF=1THEN4900:REM OK LO LLEVA
4870 SN$="TU NO TIENES EL "+IV$(7,1)
4880 GOSUB5880:GOTO4670:REM SIGUIENTE INSTRUCCION
4890 :
4900 SN$="ABRES EL LIBRO DE CODIGOS Y ENCUENTRAS LA
     PALABRA "+CD$+" ESCRITA DENTRO"
4910 GOSUB5680:GOTO4670:REM SIGUIENTE INSTRUCCION
4920 :
4930 REM ** LA INSTRUCCION ES PULSAR **
4940 IFNN$="AND"OR NN$="OR" OR NN$="NOT"THEN4970
4950 SN$="NO HAY NINGUN "+NN$:GOSUB5880:GOTO4670:REM
     SIGUIENTE INSTRUCCION
4960 :
4970 REM ** CORRECTO O INCORRECTO **
4980 IFNN$=CD$ THEN GOSUB5100:RETURN
4990 GOSUB 5010:RETURN
5000 :
5010 REM ** S/R INCORRECTO **
5020 SN$="INCORRECTO. SE ABRE UNA TRAMPA Y TE
     ENCUENTRAS DE VUELTA"
5030 SN$=SN$+" EN LA MEMORIA PRINCIPAL"
5040 GOSUB 5880:REM FORMATEAR
5050 IF RN=1 THEN P=39
5060 IF RN=2 THEN P=35
5070 IF RN=3 THEN P=29
5080 MF=1:RETURN
5090 :
5100 REM ** S/R CORRECTO **
5110 SN$="SE ABRE LA PUERTA QUE CONDUCE AL ACUMULADOR"
5120 SN$=SN$+" Y TU LA ATRAVIESAS":GOSUB 5880
5130 P=30:MF=1:RETURN

5420 REM **** BICHO AL AZAR ****
5430 SF=1
5440 SN$="UN BICHO ENORME Y ASQUEROSO SE ASOMA POR
     DETRAS DE UN CHIP"
5450 SN$=SN$+" Y SE ABALANZA SOBRE TI":GOSUB 5880
5460 :
5470 REM ** INSTRUCCIONES **
5480 PRINT:INPUT"INSTRUCCIONES";IS$
5490 GOSUB1700:GOSUB1900:REM ANALIZAR
5500 IF MF=1THENMF=0:PRINT"NO PUEDES
     MOVERTE...TODAVIA":GOTO5480
```

```
5510 IF VF=1 THEN5480:REM SIGUIENTE INSTRUCCION
5520 IF VB$="MATAR"ORVB$="LUCHAR"THEN5550
5530 PRINT"NO COMPRENDO":GOTO5480
5540 :
5550 REM ** LA INSTRUCCION ES LUCHAR/MATAR **
5560 RA=RND(TI)
5570 IFRA<0.5 THEN GOSUB5600
5580 GOSUB5670:RETURN
5590 :
5600 REM **** S/R EL BICHO TE MATA ****
5610 SN$="LUCHAS CON EL BICHO. TE ARROJA UNA LLUVIA DE"
5620 SN$=SN$+" DE ERRORES DE PROGRAMA QUE HACEN
     ESTRAGOS EN TU CEREBRO."
5630 SN$=SN$+" FINALMENTE YA NO PUEDES ABSORBER MAS Y
     TE ESTALLA LA CABEZA."
5640 GOSUB5880
5650 END
5660 :
5670 REM **** S/R TU MATAS AL BICHO ****
5680 SN$="LUCHAS CON EL BICHO Y A PESAR DE QUE LA PELEA ES
     DURA"
5690 SN$=SN$+" FINALMENTE LO VENCES Y LOGRAS
     SOBREVIVIR.":GOSUB5880
5700 RETURN
```

Complementos al BASIC

Spectrum:

En ambos programas, reemplace SN$ por S$, IS$ por T$, IV$(,) por V$(,), VB$ por B$, CD$ por C$ y NN$ por R$.

En el listado de *El bosque encantado* sustituya las siguientes líneas:

```
207 RAND
4815 IF F<>2 THEN LET S$="EL":LET
   A$=V$(F,1):GOSUB7000
4816 IF F<>2 THEN LET S$=S$+"NO
   SIRVE DE NADA":GOSUB5500:GOTO 4725
```

En el listado de *Digitaya* reemplace las líneas siguientes:

```
2750 LET RN=RND(1)
2820 LET P=INT(RND(1)*40+7)
4570 LET RN=INT(RND(1)*3+1)
4820 LET SS="TU":LET AS=VS(F,1):
     GOSUB 6500
4825 LET SS=SS+" NO SIRVE DE
     NADA":GOSUB 5880
5560 LET RA=RND(1)
```

BBC Micro:

En el listado de *El bosque encantado* reemplace estas líneas:

```
207 RND(-TIME)
4535 CR=RND(3)
```

En el listado de *Digitaya* reemplace estas líneas:

```
2750 RN=RND(1)
2820 P=RND(40)+7
4570 RN=RND(3)
5560 RA=RND(1)
```

Dibujos cicloides

Iniciamos una serie en que estudiaremos el uso del LOGO en la creación de patrones geométricos

A lo largo del curso habíamos ofrecido un método sencillo para dibujar un círculo utilizando el LOGO:

```
TO CIRCULO
  REPEAT 360 [FORWARD 1 RIGHT 1]
END
```

Esto dará una aproximación bastante acertada a lo que es un círculo. No obstante, el dibujo se realiza con una extraordinaria lentitud, si bien se lo puede acelerar un poco escondiendo la tortuga. Si en su pantalla este procedimiento no se parece a un círculo, entonces deberá volver a determinar la proporción de tamaños: habrá de experimentar una y otra vez hasta obtener un círculo y no una elipse.

Por supuesto, en realidad CIRCULO no dibuja un círculo. Dibuja un polígono de 360 lados; pero, en la mayoría de los casos, ésta es una aproximación muy aceptable. De hecho, muchas veces un polígono de 60 lados, o incluso de 30, ya es bastante adecuado y resulta mucho más rápido de dibujar. En este proyecto nuestros círculos serán polígonos de 60 o 120 lados, pero usted puede modificar esto si así lo desea. Los números mayores darán mayor detalle; con los números menores se obtendrá un dibujo más rápido.

En primer lugar, vamos a considerar qué es en realidad un *cicloide*. Imagínese un círculo que rueda a lo largo de una línea recta. Marque un punto de la circunferencia de su círculo imaginario y luego rastree el camino que sigue este punto mientras el círculo se desplaza por la línea. El camino resultante es lo que se conoce como *cicloide*. Utilizaremos esta definición como ayuda para construir un programa que nos dibuje uno.

Como una primera aproximación al cicloide, tomaremos "instantáneas" después de cada giro de 6° de un círculo que gire a lo largo de una línea que atraviese la pantalla. Cuando el círculo gira 6°, se mueve (2 × pi × radio ÷ 60) unidades a lo largo de la línea. De modo que la coordenada *x* del centro del círculo se habrá incrementado en la misma proporción (la coordenada *y*, por supuesto, no se verá afectada). Al mismo tiempo, la orientación de la línea que une el "punto de dibujo" con el centro del círculo se habrá incrementado en 6°.

La estrategia utilizada en el programa implica tres tareas sencillas:

1. mover el centro del círculo;
2. colocar la tortuga en el centro;
3. apuntarla en la dirección correcta;
4. moverla hacia adelante por la longitud del radio.

Con ello la tortuga llega hasta la siguiente posición del punto de dibujo. En este momento dibujamos un punto en la pantalla y después repetimos todo el proceso.

El procedimiento PREPARARPANTALLA es la primera llamada a procedimiento desde CICLOIDE: se encarga de algunos detalles menores necesarios para la visualización. Las principales tareas de PREPARARPANTALLA son determinar la proporción de tamaños (usted necesitará un valor diferente del nuestro) y seleccionar la modalidad NOWRAP (no desplazamiento), de modo que el programa se detenga cuando la curva se salga de la pantalla.

```
TO MOVERCENTRO
  MAKE "XCENT :XCENT + :PASO
END

TO PUNTO
  PD
  FORWARD 1
  BACK 1
  PU
END
```

Si, en lugar de tomar un punto de la circunferencia del círculo generador, rastreamos el camino efectuado por un punto dentro del círculo, entonces obtenemos lo que se conoce como un *cicloide abreviado*. Si tomamos un punto exterior al círculo, pero unido a él, obtenemos otra clase de cicloide: un *cicloide prolongado*. Para observar estos efectos podemos modificar CICLOIDE para que tome una entrada que represente la distancia del punto de dibujo respecto a la circunferencia. Los valores positivos dan cicloides abreviados; los valores negativos, cicloides prolongados:

```
TO CICLOIDE
  PREPARARPANTALLA
  MAKE "ANGULOPASO 6
  MAKE "PI 3.14
  MAKE "RADIO 15
  MAKE "CIRCUNFERENCIA 2* :PI* :RADIO
  MAKE "PASO :CIRCUNFERENCIA/(360/:ANGULO
  PASO)
  MAKE "XCENT(-150)
  CIC 0
END

TO PREPARARPANTALLA
  .ASPECT 0.93
  NOWRAP
  DRAW
  PENUP
  HT
END

TO CIC :ANG
  MOVERCENTRO
  SETXY :XCENT 0
  SETH :ANG
  FORWARD :RADIO
  PUNTO
  CIC :ANG+:ANGULOPASO
END
```

```
TO CICLOIDE :DESPL
  PREPARARPANTALLA
  MAKE "ANGULOPASO 6
  MAKE "PI 3.14
  MAKE "RADIO 15
  MAKE "CIRCUNFERENCIA 2* :PI* :RADIO
  MAKE "PASO :CIRCUNFERENCIA/(360/
  :ANGULOPASO)
  MAKE "XCENT(-150)
  MAKE "DISTANCIA :RADIO-:DESPL
  CIC 0
END

TO CIC :ANG
  MOVERCENTRO
  SETXY :XCENT 0
  SETH :ANG
  FORWARD :DISTANCIA
  PUNTO
  CIC :ANG+:ANGULOPASO
END
```

Unir los puntos

El marcar los puntos con puntos, como hemos venido haciendo hasta ahora, nos proporciona una forma sencilla de visualizar lo que está sucediendo, pero obtendríamos diagramas más atractivos si pudiéramos unir los puntos entre sí para formar una curva. El procedimiento UNIR dibuja una línea entre dos puntos:

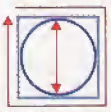
```
TO UNIR :A :B
  ESTPOS :A
  PD
  ESTPOS :B
  PU
END

TO ESTPOS :POS
  SETXY FIRST :POS LAST :POS
END
```

El procedimiento se utiliza con las coordenadas de los dos puntos dados en la llamada. Por ejemplo, una llamada posible es UNIR [12 34][67 89]. En nuestro programa de cicloides, deberemos llevar un registro de la posición antigua del punto, y después unirlo con la posición actual. El resultado final de nuestro programa perfeccionado para dibujar cicloides es:

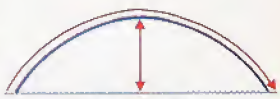
```
TO CICLOIDE DESPL
  PREPARARPANTALLA
  MAKE "ANGULOPASO 6
  MAKE "PI 3.14
  MAKE "RADIO 15
  MAKE "CIRCUNFERENCIA 2* :PI*
  :RADIO
  MAKE "PASO :CIRCUNFERENCIA/(360/
  :ANGULOPASO)
  MAKE "XCENT(-150)
  MAKE "DISTANCIA :RADIO -
  :DESPL
  MAKE "POSANT LIST :XCENT
  :DISTANCIA
  CIC 0
END

TO CIC :ANG
  MOVERCENTRO
  SETXY :XCENT 0
  SETH :ANG
  FORWARD :DISTANCIA
  MAKE "POSNUE POS
  UNIR :POSANT :POSNUE
  MAKE "POSANT :POSNUE
  CIC :ANG+ :ANGULOPASO
END

TO POS
  OUTPUT LIST XCOR YCOR
END
```

Quizá le interese realizar algunos experimentos con estos procedimientos. Por ejemplo, los libros de texto de matemáticas afirman que ¡la longitud de un arco de un cicloide es igual al perímetro de un cuadrado circunscrito en el círculo generador! Intente modificar los procedimientos para dibujar cicloides que le permitan comprobar este teorema.

Si posee un LOGO que incorpore sprites, una forma diferente (y mejor) de escribir el programa sería establecer el punto de dibujo como un sprite. Este procedimiento tiene una ventaja: usted siempre podría averiguar la situación del punto mediante el empleo de TELL y después XCOR e YCOR.



El cuadrado circunscrito



El arco cicloide

Complementos al LOGO

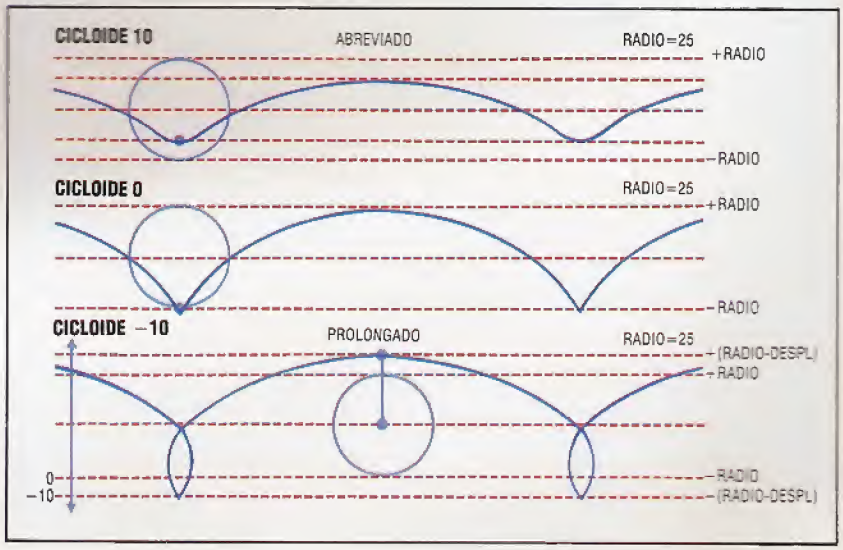
Para todas las versiones LCSI:

La sintaxis IF es diferente; por ejemplo: IF :A = 120 [STOP].
SETPOS y POS existen como primitivas.
SETXY deberá sustituirse por SETPOS (que requiere una lista como entrada).
Utilice CS por DRAW.
Para NOWRAP utilice FENCE (FENCE no existe en el LOGO Atari, de manera que emplee WINDOW y luego <BREAK> para detener el procedimiento).

Para establecer la proporción de tamaños utilice:

.SETSCR en el Atari;
SETSCRUNCH en el Apple;
SETSCRUNCH, seguida de una lista, en el Spectrum

Rodando
Un cicloide es la curva descrita por el movimiento de un punto en un radio fijo de un círculo que rueda a lo largo de una línea recta. La naturaleza de la curva difiere en función de si el punto se halla dentro, fuera o en el perímetro del círculo



Liz Dixon

Generación de hipercicloides

Nuestra investigación de patrones geométricos en LOGO comenzó con la generación de cicloides, las curvas que describen los círculos al rodar sobre una línea recta; sin embargo, con ello no se agotan, de ninguna manera, las posibilidades del círculo móvil.

En lugar de desplazarse a lo largo de una línea recta, el círculo generador podría girar dentro de otro círculo; el camino que recorrería el punto de dibujo en este caso sería un *hipercicloide*.

El problema continúa siendo básicamente el mismo; aún necesitamos mover el centro del círculo generador y luego pasar al punto de dibujo de la circunferencia. Sin embargo, ahora necesitamos llevar el registro de dos ángulos de paso. Uno (ANGULOPASO) es para calcular el camino del centro del círculo generador, y el otro (ORIENTPASO) lleva el registro de la orientación del punto dibujado con respecto al centro de este círculo. Puesto que el centro del círculo más pequeño gira, el punto dibujado gira en la dirección contraria. Se puede demostrar que los tamaños de los dos ángulos de paso están relacionados si aplicamos la fórmula que transcribimos a continuación:

```
ANGULOPASO=ORIENTPASO X
(RADIO1/RADIO2−1)
```

donde RADIO1 es el radio del círculo fijo y RADIO2 es del que gira.

El procedimiento HIPERCICLOIDE toma como entrada RADIO2, permitiéndonos, por tanto, rastrear varios hipercicloides diferentes.

```
TO HIPERCICLOIDE :RADIO2
  PREPARARPANTALLA
  MAKE "PI 3.14
  MAKE "RADIO1 60
  MAKE "DIFERENCIA :RADIO1 −
  :RADIO2
  MAKE "ORIENTPASO 6
  MAKE "CIRCUNFERENCIA 2*:PI*
  :DIFERENCIA
  MAKE "PASO:CIRCUNFERENCIA/(360/
  :ORIENTPASO)
  MAKE "ANGULOPASO :ORIENTPASO*
  (:RADIO1/:RADIO2−1)
  MAKE "CENTRO LISTA 0
  :DIFERENCIA
  MAKE "ORIENT 0
  MAKE "XCENT 0
  MAKE "POSANT LIST :XCENT
  :RADIO1
  HCIC 0
END

TO HCIC :ANG
  MOVERCENTRO2
  ESTPOS POS
  SETH :ANG
  FORWARD :RADIO2
  MAKE "POSNUE POS
  UNIR :POSANT :POSNUE
  MAKE "POSANT :POSNUE
  HCIC :ANG − :ANGULOPASO
END

TO MOVERCENTRO2
  SETXY 0 0
  SETH :ORIENT
  FORWARD :DIFERENCIA
  MAKE "CENTRO POS
  MAKE "ORIENT :ORIENT+
  :ORIENTPASO
END
```

Hay un caso especial interesante: si el radio del círculo que rueda es la mitad del radio del círculo fijo, ¡el hipercicloide se convierte en una línea recta! De este modo, el movimiento dentro de un círculo se transforma en movimiento a lo largo de una línea recta.

Tal vez desee modificar los procedimientos para averiguar lo que sucede si el punto se halla dentro del círculo o fuera del mismo.

Otro procedimiento: el "cosido de curvas"

El *cosido de curvas* es otra forma de desarrollar algunas formas interesantes a partir de círculos. Tome dos círculos concéntricos y trácelos en una gran cantidad de arcos iguales, supongamos 120. Numere los puntos y después únalos, de uno en uno, a los puntos del otro círculo de acuerdo con alguna regla sencilla; por ejemplo, x "se une a" $2x$. Los resultados pueden ser sorprendentes.

Esta actividad en realidad se puede llevar a cabo con aguja e hilo, por lo cual a menudo se alude a ella como *cosido de curvas*. También se puede realizar con lápiz y papel; pero, obviamente, preferiríamos que usted utilizara el LOGO.

Ésta es nuestra versión para un programa de cosido de curvas:

```
TO PREPARACION
  MAKE "RADIOA 80
  MAKE "RADIOB 60
  DRAW
  HT
  PENUP
  DIBUJARLO 0 0
END

TO DIBUJARLO :A :B
  IF :A = 120 THEN STOP
  UNIR PTA :A PTB :B
  MAKE "A :A+1
  MAKE "B 2* :A
  DIBUJARLO :A :B
END

TO PTA :NUM
  SETXY 0 0
  SETH :NUM*3
  FORWARD :RADIOA
  OUTPUT POS
END

TO PTB :NUM
  SETXY 0 0
  SETH :NUM*3
  FORWARD :RADIOB
  OUTPUT POS
END
```

Es posible que le interese investigar los patrones generados por otras reglas, tales como $x \rightarrow 3x$, $x \rightarrow 4x$, etc.

Compañero de clases

El Link 480Z, la más reciente creación de Research Machines, es un ordenador concebido especialmente para la enseñanza

El Link 480Z se vende en una versión normal, que es la que analizaremos aquí, y en una versión para conectar en red. A primera vista parece muy distinto de su predecesor, el 380Z. Mientras que éste se compone de una gran caja metálica de color negro que contiene el ordenador y las unidades de disco conectados mediante un cable al teclado externo, el Link 480Z posee una sólida carcasa plástica, con el teclado incorporado y las unidades de disco suministradas en una unidad externa opcional. El 480Z no es una máquina normal (mide 520 × por 330 × 80 mm), pero su aspecto estilizado es más agradable que el del funcional y más bien feo 380Z.

La máquina posee un teclado estándar QWERTY tipo máquina de escribir y las teclas son firmes, con una seguridad de tacto que las hace ideales para el tratamiento de textos. Las teclas de control, incluyendo una de salto de línea y Repeat para funciones de edición en pantalla, están situadas a la izquierda y derecha del trazado QWERTY. La única objeción que se le puede hacer al teclado es que la tecla Return quizá sea demasiado pequeña, lo que dificulta su uso.

En el lado derecho del teclado hay un grupo de teclas para control del cursor. En cada una de las esquinas del grupo hay una tecla de función programable; los fines de éstas dependen de la aplicación que se ejecute.

Una gran selección de puertas para interface, situadas en la parte posterior de la máquina, permiten conectar el ordenador a una amplia gama de periféricos. En el extremo izquierdo hay un conector hembra RF, que permite enchufar la máquina a un televisor normal. A la derecha está el botón de RESET.

El Link 480Z posee dos conectores distintos para pantalla: un conector DIN de cinco patillas, que permite conectar el ordenador a la gama de pantallas Microvitec, y, encima del mismo, un conector DIN de ocho patillas para otros tipos de pantallas TTL y RGB. Entre las puertas para pantallas y para cassette hay una interface accesoria. Ésta es una puerta de entrada/salida en serie que permite la conexión de dispositivos externos.

A la derecha de la puerta para cassette se halla la puerta de entrada/salida en paralelo, para la conexión de dispositivos en paralelo (impresoras, p. ej.). Si bien la interface no es una Centronics estándar, es compatible con ésta, lo que significa que si bien están presentes todas las líneas necesarias para un dispositivo Centronics, éstas no están situadas en el orden correcto. Un pequeño reajuste del cableado ofrecería una puerta Centronics totalmente estándar.

El Link 480Z posee asimismo un par de puertas en serie RS232 que permiten conectar la máquina en interface a dispositivos tales como impresoras en serie y las unidades de disco gemelas. Junto a las puertas en serie hay 10 interruptores DIP. El primer interruptor, señalado con una R, permite que el operador desactive el interruptor RESET. Del mismo modo, el segundo interruptor permite activar o desactivar el altavoz interno, situado debajo del teclado.

Los ocho interruptores DIP situados en el extremo izquierdo de este grupo permiten que el usuario establezca la dirección dentro de la red; éstos son leídos como un número binario para dotar al ordenador de una identificación cuando se lo conecta en red. Dado que el 480Z posee ocho de estos interruptores, permite la conexión en red de hasta 256 máquinas diferentes. El cable para la red está instalado en un enchufe hembra de video en la parte posterior del ordenador, lugar donde también hay un ventilador para mantener refrigerado el ordenador, un interruptor de on/off, un fusible y el cable de toma de corriente.

Las unidades de disco

La unidad de disco gemela MD2 está separada del ordenador. Sorprendentemente, tratándose de un micro moderno, el modelo éstandar se conecta a la máquina a través de una interface en serie y no en paralelo, y se enchufa en la segunda puerta RS232. A pesar de esto, la velocidad de transferencia es de 38,5 Kbaudios, comparable a la de muchos micros con transferencia de datos en paralelo. Las unidades gemelas utilizan los discos flexibles estándares de 5 1/4 pulgadas; éstos son de doble cara y doble densidad y están rotulados A, B, C y D. La carcasa

Chris Stevens

Aspecto elegante
El teclado sesgado y la carcasa plástica le confieren a la máquina un aire más elegante que su predecesora, el 380Z, si bien de acuerdo a las pautas modernas sigue siendo una máquina grande. Ello se debe a que dentro de la máquina hay dos niveles de placas de circuito impreso, una para las funciones principales del ordenador y otra para trabajar en red

de la unidad es del mismo plástico resistente que la del ordenador, y en funcionamiento es sumamente silenciosa en comparación con otras máquinas de oficina que se venden al doble de precio.

Detrás de las unidades hay un par de conectores RS232, uno para conectar al 480Z y el otro para conectar dispositivos "en margarita"; también poseen su propia fuente de alimentación eléctrica. El sistema de archivo en disco, que administra la transferencia de datos hacia y desde el ordenador, está instalado dentro de la carcasa de la unidad y no dentro de la propia máquina. Este empleo de unidades de disco "inteligentes" significa que el ordenador puede estar cumpliendo otras funciones mientras la gestión de los discos corre a cuenta de las propias unidades, con lo cual la memoria queda para uso del sistema.

Cuando se conecta la máquina, se le solicita al usuario que entre el BASIC ampliado basado en ROM o bien que vea el menú HELP (de ayuda). Pulsando H (de Help) se visualiza la lista de opciones disponibles basadas en ROM. Éstas se refieren fundamentalmente al sistema de entrada/salida. El operador puede optar por cargar programas del sistema ya sea desde cassette o desde disco, o cargar el sistema de red. También se pueden seleccionar la velocidad de la cassette o las opciones de impresora. Hay una opción Front Panel (básicamente, un monitor de memoria) que permite al usuario examinar y modificar los registros del procesador y las posiciones de la memoria. Relacionada con esta opción está la instrucción Jump (saltar), que permite pasar el control a una dirección de la memoria. Por ejemplo, la instrucción J103 le pasa el control al vector de "arranque en caliente".

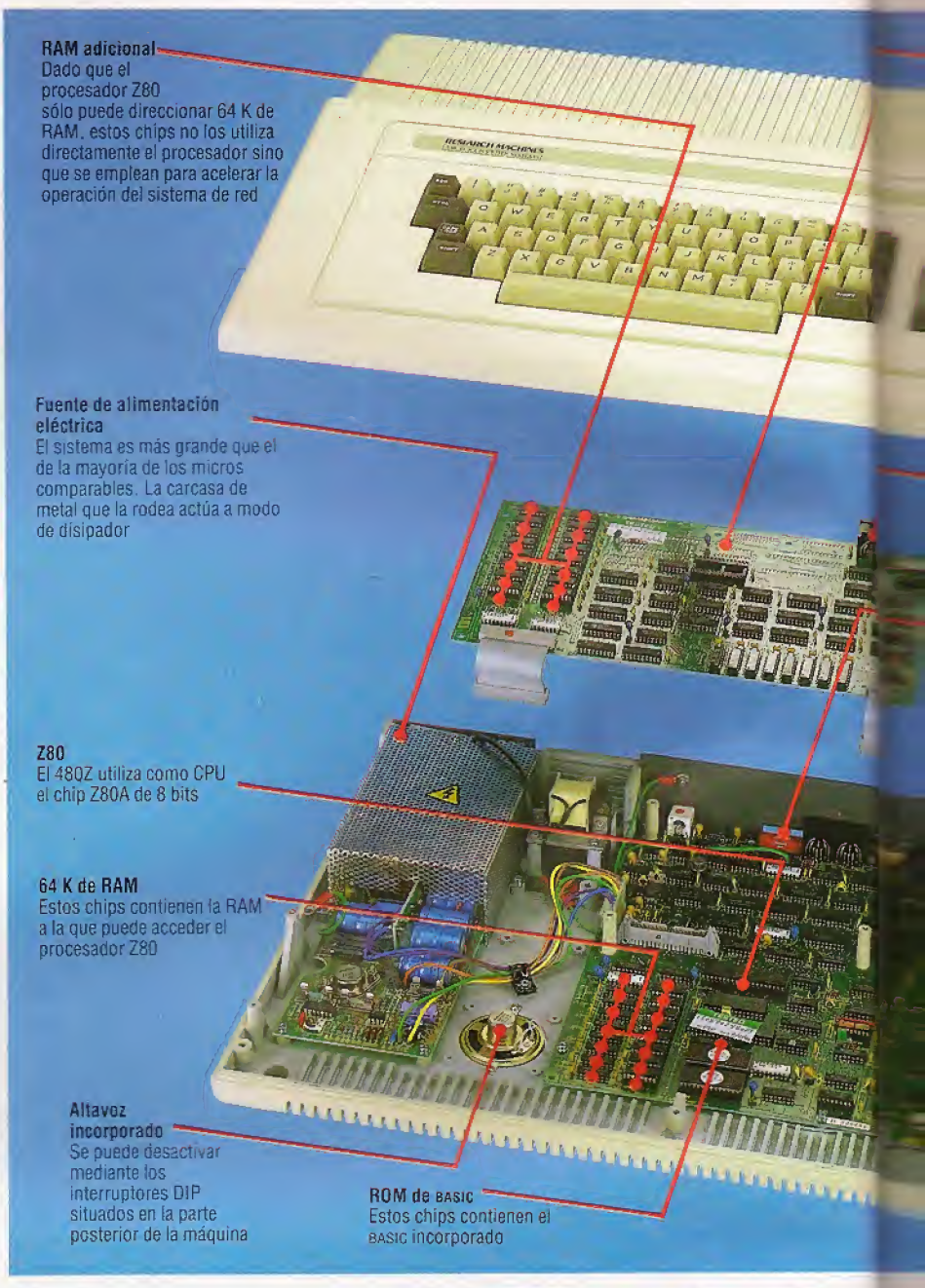
Resolución de pantalla

El 480Z dispone de numerosas modalidades de resolución de pantalla. Éstas van desde la pantalla para textos de 80 por 25 hasta la visualización de resolución ultra alta, de 640 por 192 (si bien ésta sólo admite dos colores en pantalla). Hay, además, tres modalidades de color que, en resolución media, soportan la gama completa de 16 colores.

Al igual que el 380Z, el Link 480Z utiliza el microprocesador Z80, que le permite ejecutar una amplia gama de software disponible, incluyendo, por supuesto, el sistema operativo CP/M. La disponibilidad de software quizá haya sido la causa primordial por la cual Research Machines decidiera continuar con este chip en vez de adoptar un procesador más moderno. Si bien la empresa sostiene que el 380Z y el 480Z tienen compatibilidad de software, ciertos programas llamados desde BASIC generan un error de disco cuando el 480Z intenta leer el disco.

En el interior de la máquina se ha dejado espacio para la adición de chips extras. Aunque esta facilidad no es comparable con el 380Z, que está diseñado de modo que las placas extras se puedan instalar con suma facilidad, sí significa que se pueden añadir aplicaciones basadas en ROM, como puede ser el convertidor de digital a analógico para permitir la conexión de la puerta accesoria a un dispositivo analógico.

Con el ordenador viene un disco de sistema que incluye numerosos programas de demostración, una versión de BASIC con instrucciones para gestión de disco y el sistema operativo CP/M.



Unidades gemelas

Cada unidad es de doble cara, dando un total de cuatro caras a las cuales acceder. Éstas, siguiendo la convención CP/M, están rotuladas como A, B, C y D. La unidad acepta discos de doble densidad, lo cual le permite a la máquina almacenar el doble de información en cada cara de los discos. Además, Research Machines ha producido una unidad de densidad cuádruple

El diseño del Link 480Z parece sugerir que la intención de Research Machines ha sido desarrollar un ordenador para satisfacer a los usuarios escolares que, no necesitando de la sólida flexibilidad del 380Z, sí requieren una máquina adaptable de propósito general. En este sentido no cabe duda de que la empresa ha acertado. Sin embargo, es una decepción encontrarse con que la máquina introduce tan pocas innovaciones, y su enorme tamaño continúa siendo un misterio. Al mantener el chip Z80, la empresa decidió que la disponibilidad de software a un precio asequible sobrepasa con creces cualquier otra ventaja que hubiera podido obtener si escogía un procesador más moderno de 16 bits. Por otra parte, los lotes de software de los paquetes escolares son una ganga. No obstante, en una época en la que el IBM-PC se está convirtiendo en el estándar para las máquinas de oficina, la elección del Z80, en lugar del chip Intel 8088, parece traslucir una cierta estrechez de miras.

Paquete escolar

El lote de software del paquete escolar que se entrega con el 480Z ofrece unas prestaciones excelentes. Se suministran 12 paquetes basados en disco, todos de gran valor educativo.

En el lote se incluyen cuatro lenguajes. El SBAS es una versión de BASIC estructurado y está considerado como una implementación excelente. Contiene una amplia gama de estructuras de control para el flujo del programa, incluyendo WHILE...ENDWHILE, CASE... ENDCASE e IF...ENDIF. Dispone asimismo de procedimientos y de variables globales y locales. La máquina también soporta una amplia implementación del PASCAL, que le ofrecerá al estudiante un conocimiento operativo completo de este lenguaje. La documentación que se proporciona con el lenguaje no es de fácil lectura, pero es muy detallada. También se ofrece LOGO en una versión excelente, si bien algunas instrucciones no son estándares. Por ejemplo, esta versión utiliza la instrucción BUILD (construir) en lugar de TO para crear procedimientos. Existe también una implementación parcial del LOGO, denominada ARROW. Para la programación de bajo nivel, el ZASM proporciona el lenguaje ensamblador Z80 para el desarrollo de programas en lenguaje máquina.

Como ayuda al desarrollo de la habilidad de teclado y proceso de textos se proporcionan cuatro programas diferentes. El Touch'n'Go está diseñado para desarrollar la destreza en la mecanografía al tacto. WORD es un curso de proceso de textos para principiantes, concebido para enseñar al alumno los principios y las técnicas que se emplean en el tratamiento de textos. Para una implementación completa, en el lote también se incluye el WordStar. TXED es un editor de textos que se puede utilizar ya sea para tratamiento de textos o bien para desarrollo de programas.

Quest-D es una base de datos diseñada para enseñar los principios del almacenamiento y la recuperación de datos. Más especializado es el SIR, destinado a catalogar los recursos de la biblioteca de la escuela y para enseñar las técnicas de las funciones de gestión de una biblioteca.

Por último, Telesoftware es un sistema de videotexto particularmente útil al utilizarlo junto con las capacidades de red del 480Z.

Chris Stevens

LINK 480Z

DIMENSIONES
520 x 330 x 80 mm

CPU
Z80, operando a 4 MHz

MEMORIA
64 K de RAM

PANTALLA
Texto: 80 x 25 caracteres
Resolución media: 160 x 192 con 16 colores
Alta resolución: 320 x 192 con cuatro colores
Resolución ultra alta: 640 x 192 con dos colores

INTERFACES
Puerta RF, pantalla, RGB/TTL, puerta accesoria, cassette, puerta en paralelo, dos puertas en serie, red, enchufe de video para conexión en red

LENGUAJES DISPONIBLES
BASIC, LOGO y PASCAL

TECLADO
65 teclas, incluyendo teclas de función y de control del cursor

DOCUMENTACION
Los manuales son exhaustivos y contienen toda la información necesaria tanto para el principiante como para el usuario avanzado. Sin embargo resulta difícil hallar parte de la información

VENTAJAS
El Link 480Z se ha desarrollado para su uso en la clase y en muchos sentidos es ideal para las escuelas. La capacidad para ejecutar CP/M y la riqueza de software disponible, junto con sus capacidades para conexión en red, le confieren gran versatilidad. El ordenador está bien construido y puede servir durante muchos años

DESVENTAJAS
Se trata de un ordenador anticuado que, al compararlo con máquinas que ofrecen capacidades similares, parece tener un precio excesivo

Paquetes eficientes

Examinemos cuatro programas destinados a microordenadores personales: "Micro Swift", "Practicalc II", "PS" y "Vizastar"

Micro Swift, Practicalc II, PS y *Vizastar* pertenecen a la nueva clase de paquetes perfeccionados basados en hoja electrónica, que evidentemente están inspirados en el paquete integrado *Lotus 1-2-3* y en su sucesor, el *Symphony*. Pero mientras que el *1-2-3* y el *Symphony* se escribieron para el IBM-PC y máquinas compatibles (para ejecutar el *1-2-3* se requieren 296 K de memoria para el usuario, y el *Symphony* exige un mínimo de 320 K), los nuevos paquetes están diseñados para micros personales. En muchos sentidos, los cuatro paquetes que vamos a analizar han hecho milagros para comprimir muchas de las características disponibles en paquetes más grandes en los aproximadamente 30 K de memoria de que dispone el usuario de micros tales como el Commodore 64.

Sin embargo, por el momento estos paquetes "miniSymphony" sólo pueden ofrecer dos de las cuatro opciones que hacen que los paquetes más potentes (y más caros) sean tan atractivos. Dadas las actuales limitaciones de hardware, intentar incorporar las cuatro opciones (hoja electrónica, base de datos, tratamiento de textos y posibilidad de programación) sin duda alguna habría exigido el tipo de concesiones que han hecho que el software basado en ROM Three-Plus-One del Commodore Plus/4 se haya convertido en algo decepcionante.

Ventajas relativas

Consideremos algunas de las opciones que ofrecen estos cuatro programas con el fin de comparar sus ventajas relativas. El *PS*, el *Micro Swift* y el *Vizastar* son, en mayor o menor grado, programables. Ésta es una facilidad sumamente valiosa, puesto que permite al usuario automatizar funciones cuya ejecución, de lo contrario, requeriría muchas pulsaciones de teclas; ello se consigue de la misma forma en que se emplean las macros de teclado en el *Lotus 1-2-3*.. Los tres programas lo hacen de forma diferente; analizaremos estos enfoques uno por uno.

En el paquete *PS* los módulos se programan empleando instrucciones familiares del BASIC. Estos módulos se guardan luego pulsando <f3> y se ejecutan utilizando <U>, o bien pueden ejecutarse automáticamente después de la carga si son salvados en disco con un punto y aparte después del nombre del programa. El paquete posee toda una gama de útiles facilidades de programación: por ejemplo, puede bifurcar a una subrutina de un programa desde una fórmula dentro de una celda simplemente insertando dentro de la fórmula la instrucción GOSUB. Pueden definirse funciones utilizando la función FN, y el programa también dispone de la facilidad de pasar series de caracteres, fila y columna, y valores numéricos.

El *Micro Swift* se puede programar colocando una lista de instrucciones en la columna Z; la primera instrucción da el nombre del programa, precedido por un signo numérico (#) y la última línea contiene la instrucción @ QUIT. Veamos un ejemplo sencillo:

```
Z1  # SUM              (sumar)
Z2  @ SUM(A1,A3)       (sumar)
Z3  @ ASSIGN(Z2,A4)    (asignar)
Z4  @ QUIT             (salir)
```

Este programa sumará los valores contenidos en las celdas A1, A2 y A3, y después le asignará el resultado, que ahora se halla en la celda Z2, a la celda A4. El programa es llamado mediante la instrucción # SUM.

De todos los paquetes mencionados aquí, quizá el más sencillo sea el *Vizastar*, puesto que las instrucciones consisten en las letras iniciales que se pulsarían para ejecutarlas de forma manual. Por lo tanto, para utilizar una base de datos específica, se pulsaría la tecla CBM seguida de D(ata: datos), U(se: usar), D(atabase: base de datos) y el nombre de la base de datos. Por último, pulsaría <RETURN>. Para programar, se utiliza el signo de barra (/) en lugar de la tecla CBM, de modo que /DUDnombre-[RET] ejecutará la acción si se pulsa <F8>. Las teclas de función y edición se programan pulsando <CRTL> más la tecla apropiada, y esta letra se imprime cuando se utiliza la función en un programa. No obstante, cuando se programan de esta manera las teclas del cursor, éstas se imprimen como [up] (arriba), [down] (abajo), [left] (izquierda) o [right] (derecha).

La base de datos del *Vizastar* es una implementación muy potente que en realidad utiliza una sección de la hoja que no está disponible para el usuario (filas de la 1 000 en adelante) para almacenar formatos de registros. Cada registro puede constar de hasta nueve pantallas, y éstos pueden ser accedidos mediante las instrucciones Key o Next, Prior, First, Last o Current (utilizando cada una la letra inicial de un menú de instrucciones). Asimismo, los registros se pueden Add (sumar), Replace (modificar) o Delete (eliminar).

Los campos tienen nombres de letras, empezando con la A y terminando con BK, que aluden a las columnas de ese nombre en la hoja electrónica. Por consiguiente, a modo de ejemplo, los criterios de búsqueda se pueden preparar en una línea vacía de la hoja electrónica. A siempre es el campo clave, es decir, el campo en el cual se clasifican los datos.

Practicalc II es una hoja electrónica que utiliza una facilidad de "etiqueta larga", que permite disponer texto "a caballo" de varias celdas. Esta facilidad permite que el programa opere como un procesador de texto con una longitud de línea máxima de 100 caracteres. Esta opción dispone de la mayoría de las facilidades más comunes de tratamiento de textos, incluyendo desplazamiento de palabras,

Transporte informatizado

La mayoría de las flotas de camiones en Gran Bretaña están compuestas por unos cinco vehículos. Pero la mayor parte de los paquetes para ordenador disponibles para administrarlas están diseñados para una cantidad mayor de camiones. El paquete para administración de flotas *MEM Computing*, por ejemplo, puede manipular flotas de 1 000 o más vehículos.

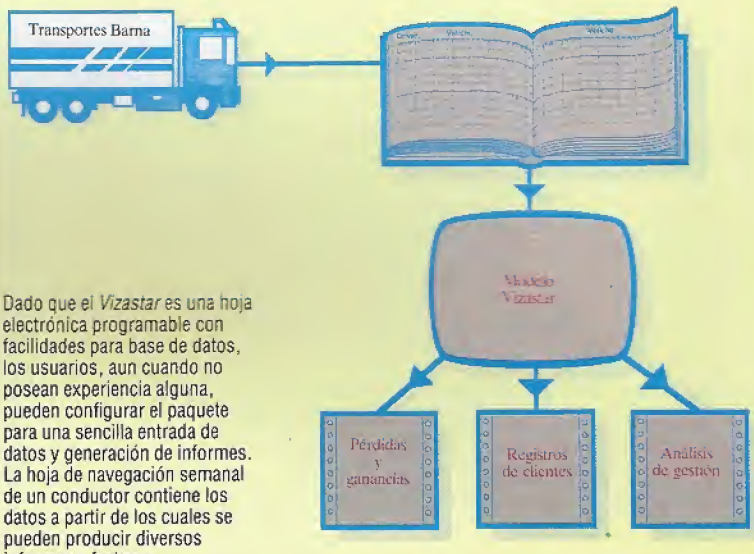
No es sorprendente que, por este motivo, sean pocos los propietarios de flotas pequeñas que hayan optado por informatizar sus operaciones, tal como descubrió Terry Palmer, consejero de transportes británico, cuando llevó a cabo un estudio de mercado patrocinado por el Science and Engineering Research Council. A consecuencia del mismo, Palmer se abocó a la tarea de crear un sistema que tuviera un sentido más comercial. Aunque comenzó a desarrollarlo utilizando el *Lotus 1-2-3*, acabó por hacerlo caber en la memoria de un Commodore 64 empleando el *Vizastar*, una combinación de hoja electrónica y base de datos programable. El costo total, según ha calculado Palmer, ha ascendido a 1 000 libras (unas 200 000 ptas), incluyendo software y hardware: alrededor de la quinta parte del costo total de los sistemas más grandes.

La investigación la realizó Palmer como parte de un proyecto que está llevando a cabo en asociación con el Polytechnic de Central London, para ver si a los pequeños camioneros la información que les proporciona tal sistema les resulta útil y si estarían preparados para invertir en la misma. Comenzó con la hoja de navegación que usan todos los transportistas y terminó con formularios de informes en los cuales los camioneros registran todos los trabajos efectuados, sus recorridos, destinos, gastos de viaje, costo de combustible, gastos en efectivo y costos operativos. Al final de cada semana, los datos de las hojas se transfieren a la hoja electrónica.

Al finalizar la entrada de datos, la hoja ya ha calculado si se han obtenido pérdidas o beneficios, y produce un análisis completo de la semana comercial. Las semanas se pueden consolidar posteriormente en análisis mensuales, y los meses en un total anual.

Dado que el *Vizastar* también trata una parte de la hoja como si fuera una base de datos, la grabación de registros de disco y su recuperación es exactamente igual que en los programas exclusivos de bases de datos (utilizando un campo de clave o posibilitando la revisión de la lista mediante el empleo de las instrucciones Next, Prior o Current, First o Last), con lo que se puede llevar un registro permanente de clientes. También en la hoja se pueden incluir anotaciones.

En marcha



Dado que el *Vizastar* es una hoja electrónica programable con facilidades para base de datos, los usuarios, aun cuando no posean experiencia alguna, pueden configurar el paquete para una sencilla entrada de datos y generación de informes. La hoja de navegación semanal de un conductor contiene los datos a partir de los cuales se pueden producir diversos informes y facturas

Ian McKinnell

desplazamiento de bloques, inserción y supresión.

También se puede cargar una hoja electrónica en una parte de un documento. La hoja electrónica seguiría estando "activa" (lo que significa que sus fórmulas, valores u otros contenidos pueden ser modificados en el documento principal, sin, por supuesto, afectar a la hoja en disco).

Si bien las limitaciones de memoria imponen que se pueda acceder sólo a un par de opciones dentro de cualquiera de estos programas, los cuatro pueden acceder a archivos de tratamiento de textos o base de datos producidos mediante otros programas editados por la misma empresa. Por ejemplo, el *Vizastar* puede manipular archivos de tratamiento de textos generados mediante el *Vizawrite*; el *Micro Swift* puede acceder a archivos de base de datos producidos por el *Micro Magpie*, y el *Practicalc* y el *PS* pueden utilizar archivos del *Practifile* de Practicorp. En realidad, puesto que todos utilizan formatos secuenciales, pueden acceder a archivos mutuos, así como a programas absolutamente dispares, como el procesador de textos *Easy script*. Aunque no puede decirse que esto sea exactamente una integración completa de software, sí nos aproxima mucho a ella.

Micro Swift: Para el Commodore 64
Editado por: Audiogenic, PO Box 88, Reading, Berks., Gran Bretaña
Formato: Disco

Practicalc II: Para el Apple II de 48 K, el BBC Micro y el Commodore 64
Editado por: Practicorp, Goddard Road, Whitehouse Ind Est, Ipswich, Suffolk IP1 5NP, Gran Bretaña
Formato: Disco

PS: Para el Commodore 64
Editado por: Practicorp, Goddard Road, Whitehouse Ind Est, Ipswich, Suffolk IP1 5NP, Gran Bretaña
Formato: Disco

Vizastar: Para el Commodore 64
Editado por: Viza Software, 9 Mansion Row, Brompton, Gillingham, Kent ME7 5SE, Gran Bretaña
Formato: Disco con cartucho de 4 K

Control exacto

Llegados a este punto, estudiaremos el calibrado del robot para lograr un preciso control en sus movimientos

Los motores paso a paso son ideales para el control mediante dispositivos digitales, dado que giran un intervalo preciso cada vez que reciben un impulso. Para establecer una relación entre el control digital del motor paso a paso y el mundo real de distancias y ángulos, hemos de llevar a cabo algunos experimentos iniciales con nuestro robot. Estos nos permitirán determinar la cantidad de impulsos necesarios para moverlo a través de varios ángulos y distancias. Luego de realizar estos experimentos estaremos en condiciones de determinar las relaciones promedio de impulso/distancia e impulso/ángulo, que deberemos entrar en los programas a modo de constantes. En futuros capítulos diseñaremos software que, unido a otras aplicaciones, permitirá al robot sondear y construir representaciones digitales de objetos sólidos. Para conseguir que funcione con precisión necesitaremos los valores de las relaciones obtenidas en los experimentos realizados con anterioridad en este apartado.

Calibrado lineal

Podemos hacer una hipótesis de la relación impulso/distancia de nuestro robot valiéndonos de la matemática elemental. Puesto que un impulso produce en los motores un giro de 7,5°, colocar la salida del motor en un coeficiente de reducción de 25:2 significará que un impulso producirá un giro de 7,5 × 2/25 = 0,6° en el eje. Puesto que la rueda Lego tiene un radio de 30 mm, el movimiento lineal por impulso se puede calcular del siguiente modo: 1 impulso produce un movimiento de 0,6/360 × 2 × *pi* × 30 mm. Descomponiendo esta expresión encontramos que 1 impulso produce un movimiento de 0,1 × *pi* mm. La inversión de esta cifra nos da un coeficiente teórico de impulso/distancia: coeficiente i/d=3,183.

El programa de calibrado que ofrecemos a continuación le permitirá efectuar varias pruebas con su robot a través de diversas distancias. A cada ejecución se visualizan en la pantalla la cantidad de impulsos y las distancias teóricas que se habrán recorrido. Utilizando dos reglas de 30 cm dispuestas una a continuación de otra, se puede registrar la distancia real recorrida en cada prueba. El programa visualiza luego una tabla de la cantidad de impulsos, las distancias reales registradas y los cálculos teóricos. También se calcula un coeficiente i/d promedio. Esta cifra es importante, de modo que conviene apuntarla por separado. La muestra de salida de este programa indica que nuestro robot prototipo tiende a recorrer, para un número dado de impulsos, una distancia ligeramente mayor de lo que sugeriría la teoría. La importancia de este ejercicio radica en que usted halle el coeficiente i/d para su robot y lo utilice en futuros programas.

```
 10 REM **** CALIBRADO BBC ****
 20 RDD=&FE62:REGDAT=&FE60
 30 ?RDD=15:REM LINEAS 0-3 SALIDA
 50 adelante=4:atras=2:DIM MD(12)
 60 FOR CC=500 TO 1700 STEP 100
 70 ?REGDAT=0
 80 ?REGDAT=(?REGDAT OR 1) OR adelante
 90 PRINT CC,INT(CC*PI)/10
100 A$=GET$
110 FOR I=1 TO CC
120 PROCimpulso
130 NEXT I
140 INPUT"DISTANCIA MEDIDA EN MM";MD((CC-500)/100)
150 NEXT CC
160 ?REGDAT=0:T=0
180 PRINT"     IMPULSOS","  MEDIDOS", "  TEORICOS."
190 PRINT
200 FOR CC=500 TO 1700 STEP 100
210 PRINT CC,MD(CC-500)/100),INT(CC*PI)/10
220 T=T+CC/MD((CC-500)/100)
230 NEXT CC
240 PRINT:PRINT " COEFICIENTE IMPULSO/DISTANCIA:",T/12
260 END
270 DEF PROCimpulso
280 ?REGDAT=(?REGDAT OR 8)
290 ?REGDAT=(?REGDAT AND 247)
300 ENDPROC
```

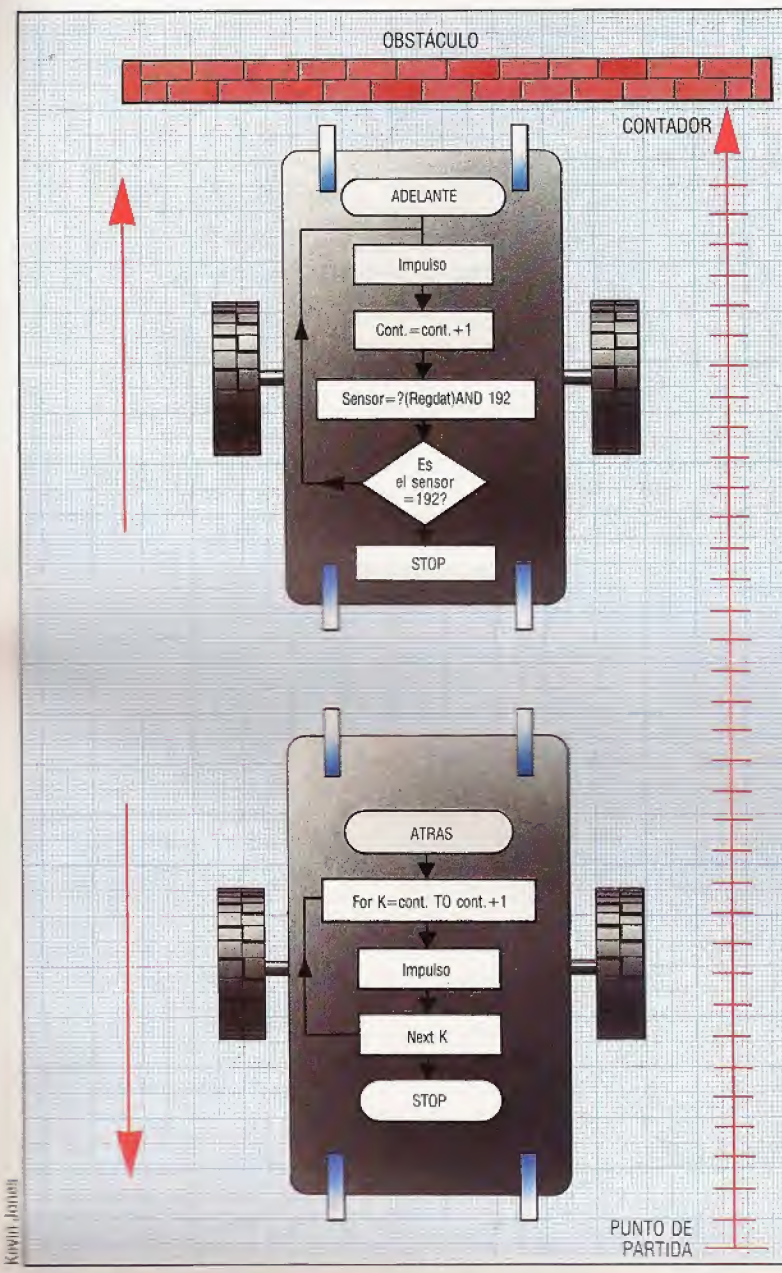
```
 10 REM **** CALIBRADO CBM 64 ****
 20 RDD=56579:REGDAT=56577
 30 POKE RDD,15:REM LINEAS 0-3 SALIDA
 50 AD=4:AT=2:DIM MD(12)
 60 FOR CC=500 TO 1700 STEP 100
 70 POKE REGDAT,0
 80 POKE REGDAT,(PEEK(REGDAT)OR 1)OR AD
 90 PRINT CC,INT(CC*π)/10
100 GET A$:IF A$="" THEN 100
110 FOR I=1 TO CC
120 GOSUB 270:REM IMPULSO
130 NEXT I
140 INPUT"DISTANCIA MEDIDA EN MM";MD((CC-500)/100)
150 NEXT CC
160 POKE REGDAT,0:T=0
170 REM ** LINEAS 180-260 IGUALES QUE VERSION BBC **
175 REM ** PERO REEMPLAZAR PI POR π EN LA LINEA 210 **
270 REM **** S/R IMPULSO ****
280 POKE REGDAT,PEEK(REGDAT)OR 8
290 POKE REGDAT,PEEK(REGDAT)AND 247
300 RETURN
```

Calibrado angular

Podemos calcular un coeficiente impulso/ángulo de este modo: si la distancia entre ejes es de 140 mm, la circunferencia del círculo de giro = 140×*pi*. Un impulso produce un giro de 360×0,1×*pi*/(140×*pi*) grados y, por tanto, el coeficiente i/a es 3,846.

Uno de los principales problemas que implica el calibrado angular es la medición exacta de los ángulos. Dado que en la mayor parte de las aplicaciones el robot girará 90° (o múltiplos de 90), nuestro coeficiente teórico de i/a nos indica que se requerirían 346 impulsos para un giro en ángulo recto.

Marque en un trozo de papel un par de líneas perpendiculares. En una de ellas haga dos pequeñas marcas a cada lado del punto donde se intersectan ambas, para designar los puntos de partida de las ruedas del robot. Ejecute el siguiente programa para hacer que el robot gire 90°. El bucle FOR-...NEXT de la línea 70 dicta la cantidad de impulsos que se le pasan a los motores. La cifra 371 es el valor experimental requerido para que nuestro robot prototipo gire 90°. Edite el programa, alterando el límite superior de este bucle, hasta que las ruedas de su robot queden alineadas exactamente con la otra línea perpendicular dibujada sobre el



IMPULSOS	MEDIDOS	TEÓRICOS
500	160	157
600	195	188,4
700	225	219,9
800	258	251,3
900	293	282,7
1000	324	214,1
1100	352	345,5
1200	390	376,9
1300	421	408,4
1400	452	439,8
1500	488	471,2
1600	522	502,6
1700	553	534

COEFICIENTE IMPULSOS DISTANCIA: 3,34767511

Tabla teórica
El programa de calibrado del robot produce esta tabla, que se debe verificar para asegurar que cada distancia medida guarde una relación razonable con la distancia teórica correspondiente. Se calcula un coeficiente i/d global como el promedio de los coeficientes de i/d obtenidos en las 12 pruebas. Este número se debe conservar, porque será necesario en futuros programas

Adelante y atrás

El movimiento hacia adelante se efectúa estableciendo los bits de dirección hacia adelante de los activadores del motor paso a paso y enviándoles un impulso a los motores. Se va incrementando un contador de impulsos y se comprueban los bits de sensores del registro de datos en busca de una entrada de los sensores de colisión. Este proceso se repite hasta que se detecta una colisión, momento en el cual se invierten los bits de dirección del motor y un bucle FOR...NEXT envía los impulsos necesarios para devolver al robot hasta su punto de partida. El movimiento del robot hacia adelante es más lento y ligeramente más torpe que el de regreso

papel. Se pueden efectuar otras comprobaciones. Reemplace la dirección "derecha" (DE) por "izquierda" (IZ) en la línea 60 y asegúrese de que el robot también gira 90° en sentido antihorario. De duplicar el valor superior del bucle FOR...NEXT, se produciría un giro de 180°. Compruebe que las ruedas terminan en los mismos puntos en los cuales empezaron. De no ser así, es necesario un ligero ajuste de las ruedas para asegurar que estén situadas simétricamente a cada lado del eje central. Cuando esté satisfecho con la posición de las ruedas, marque sus posiciones en el eje y péguelas en su sitio.

```
10 REM **** GIRO CBM 64 ****
20 RDD=56579:REGDAT=56577
30 POKE RDD,15:REM LINEAS 0-3 SALIDA
40 IZ=6:DE=0
50 POKE REGDAT,0
60 POKE REGDAT,(PEEK(REGDAT)OR 1)OR DE
70 FOR I=1 TO 371:GOSUB 90:NEXT I
80 POKE REGDAT,0:END
90 REM **** S/R IMPULSO ****
100 POKE REGDAT,PEEK(REGDAT)OR 8
110 POKE REGDAT,PEEK(REGDAT)AND 247
120 RETURN
```

```
10 REM **** GIRO BBC ****
20 RDD=&FE62:REGDAT=&FE60
30 ?RDD=15:REM LINEAS 0-3 SALIDA
40 izquierda=6:derecha=0
50 ?REGDAT=0
60 ?REGDAT=(?REGDAT OR 1)OR derecha
70 FOR I=1 TO 371:PROCimpulso:NEXT
80 ?REGDAT=0:END
90 DEF PROCimpulso
100 ?REGDAT=(?REGDAT OR 8)
110 ?REGDAT=(?REGDAT AND 247)
120 ENDPROC
```

Ahora que hemos incorporado a nuestro robot sensores de microinterruptor podemos escribir software que utilice la salida de la puerta para el usuario para controlar el robot, y la entrada para monitorizar las actividades externas a través de los sensores del robot. El siguiente y sencillo programa envía al robot hacia adelante hasta encontrar un objeto, momento en el cual el robot retrocede exactamente hasta su posición de partida. La lógica del programa se puede describir del siguiente modo:

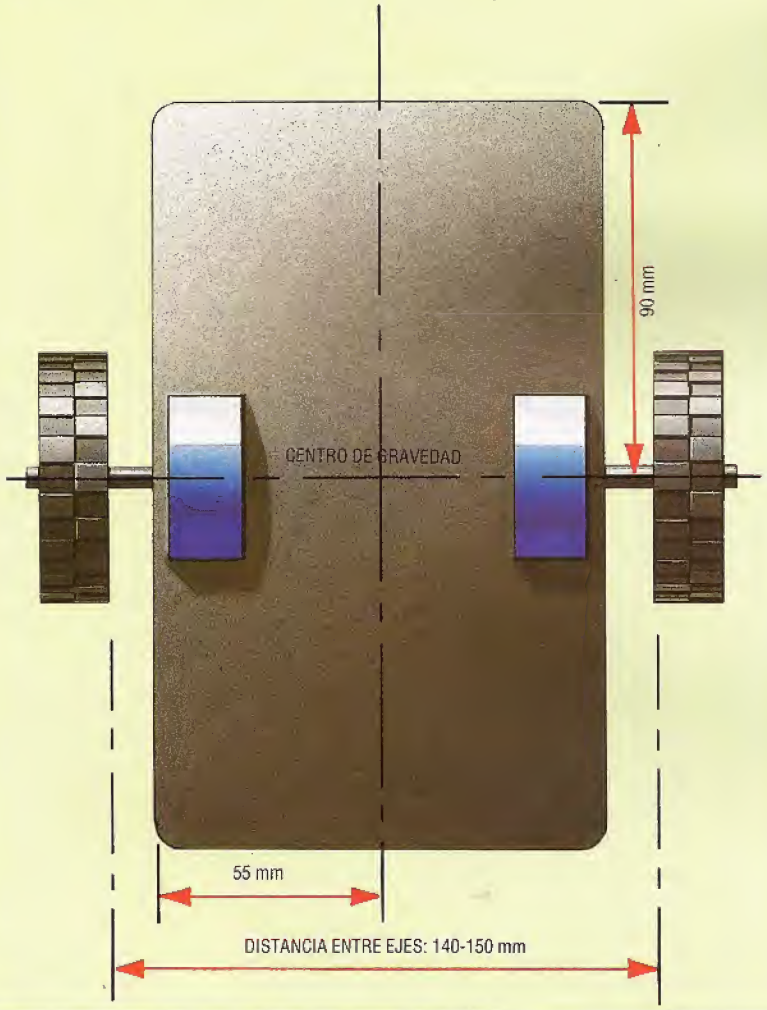
1. Establecer el registro de dirección de datos en 15. Con ello se establecen los bits 0–3 como salida y los bits 4-7 como entrada.

2. Establecer la dirección del robot hacia adelante.
3. Impulsar los motores hasta poner bajo el bit 6 o el bit 7, llevando la cuenta de la cantidad de impulsos efectuados.
4. Establecer la dirección del robot hacia atrás.
5. Impulsar los motores las veces que indique el "contador".
6. Establecer el reg. de datos en 0 y terminar.

```
 10 REM **** PARACHOQUES CBM ****
 20 RDD=56579:REGDAT=56577
 30 POKE RDD,15:REM LINEAS 0-3 SALIDA
 40 AD=4:AT=2
 50 POKE REGDAT,(PEEK(REGDAT)OR 1)OR AD
 60 REM **** IMPULSO ADELANTE ****
 65 CC=0
 70 GOSUB 1000:CC=CC+1:REM IMPULSO
 80 IF (PEEK(REGDAT)AND 192)=192 THEN 70
 90 REM **** REGRESAR A PARTIDA ****
 95 POKE REGDAT,(PEEK(REGDAT)AND 1)OR AT
100 FOR I=1 TO CC
110 GOSUB 1000:REM IMPULSO
120 NEXT I
130 POKE REGDAT,0:END
```

Ajustes preliminares

Antes de comenzar el proceso de calibrado es necesario efectuar algunos ajustes. Primero, con el robot patas arriba, es importante situar correctamente el eje central sobre el cual girará éste. Esto se puede hacer mediante las mediciones que se indican. Marque el centro del eje con un pequeño rasguño o con un rotulador indeleble. Mida la distancia entre las caras interiores de las ruedas (debe estar entre 140 y 150 mm). Es importante, para que el robot pivote correctamente, que cada rueda sea equidistante respecto al punto central del eje que hemos marcado. Deslice suavemente las ruedas a lo largo de sus ejes hasta lograrlo



Kevin Jones

```
1000 REM **** S/R IMPULSO ****
1010 POKE REGDAT,(PEEK(REGDAT)OR 8)
1020 POKE REGDAT,(PEEK(REGDAT)AND 247)
1030 RETURN
```

```
 10 REM **** PARACHOQUES BBC ****
 20 RDD&FE62:REGDAT=&FE60
 30 ?RDD=15:REM LINEAS 0-3 SALIDA
 40 adelante=4:atras=2
 50 ?REGDAT=(?REGDAT OR 1) OR adelante
 60 REM **** IMPULSO ADELANTE ****
 65 contador=0
 70 REPEAT:PROCimpulso:contador=contador+1
 80 UNTIL (?REGDAT AND 192)<>192
 90 REM **** REGRESAR A PARTIDA ****
 95 ?REGDAT=(?REGDAT AND 1)OR atrás
100 FOR I=1 TO contador
110 PROCimpulso
120 NEXT I
130 ?REGDAT=0:END
1000 DEF PROCimpulso
1010 ?REGDAT=(?REGDAT OR 8)
1020 ?REGDAT=(?REGDAT AND 247)
1030 ENDPROC
```

En este programa hemos designado al par de interruptores hacia adelante como el par que está más alejado de los conectores de parche de la tapa del robot, y hemos conectado estos dos microinterruptores a los bits 6 y 7, utilizando dos cables entre los dos pares de conectores situados más a la derecha de la tapa, el rojo y el azul. En el futuro daremos siempre por sentado que el enchufe D está aún más adelante que el sistema de conectores de parche. Si al ejecutar este programa se encuentra con que su robot avanza primero hacia atrás (de acuerdo a esta convención), quite la tapa y déle la vuelta.

De los cuatro bits bajos del registro de datos que controlan la operación del motor, el bit 0 es el bit reposo (establecido normalmente en uno), los bits 2 y 3 son los controladores de dirección para los motores derecho e izquierdo, y el bit 3 impulsa simultáneamente ambos motores, haciéndolos girar un paso cada vez que el bit 3 sufre una transición de bajo a alto. El empleo de los operadores lógicos AND y OR permite encender y apagar bits individuales sin afectar a los otros bits del registro. Dado que los cuatro bits superiores han sido establecidos como entradas por el registro de dirección de datos, se suelen mantener altos. Cuando un microinterruptor se cierra, el bit correspondiente del registro de datos pasa a bajo. Normalmente, los bits 6 y 7, si estuvieran establecidos para entrada, tendrían el valor 192 (128+64). El bucle repetitivo que envía el robot hacia adelante en las líneas 70-80, termina con la condición de que estos dos bits ya no posean un valor de 192. Esto puede suceder si se cierra un microinterruptor (o ambos). Si se lleva un contador de la cantidad de impulsos enviados a los motores en el período intermedio, entonces el robot puede retroceder exactamente hasta su punto de partida mediante la alteración de los bits de dirección del motor e impulsando a los motores la cantidad apropiada de veces. El paso de 7,5° de los motores se traduce en un movimiento de menos de 1 mm para la rueda; en consecuencia, podemos controlar muy fácilmente la posición del robot.

Por último, es interesante destacar que el robot se mueve hacia adelante más despacio que cuando se mueve hacia atrás. Aquí nos vemos limitados por la velocidad del BASIC. El tiempo entre impulsos en el bucle que envía al robot hacia adelante es mayor que el que lo hace retroceder, porque en el primer bucle se ha de llevar a cabo un trabajo adicional, como llevar el contador y comprobar la colisión, lo que no sucede en el segundo bucle.

Ahora haremos un alto en el proyecto del robot para darle tiempo para completar el ensamblaje.

Entreacto

Finalizamos nuestra introducción al sistema operativo del BBC concentrándonos en la utilización de vectores y examinando la interacción con el ordenador mediante el teclado y la unidad de representación visual (VDU)

Se dice que la mayoría de las rutinas del OS del BBC están *vectorizadas*. El OS, cuando se le instruye para que llame a la rutina OSCLI, lo primero que hace es llamar a una rutina en la dirección $FFF7. Es ésta la rutina que llama a su vez a la OSCLI, pero no directamente sino que halla la dirección donde se encuentra la OSCLI por medio del contenido de dos bytes de memoria que se encuentran en la página 2 de la RAM. Estos dos bytes son llamados *vector*: el byte inferior de la dirección en cuestión se encuentra en el byte de inferior numeración de dicho vector, y el byte superior de la dirección se halla en el byte de numeración superior. Así, para OSCLI, que se vectoriza mediante las posiciones $208 y $209, el byte inferior de su dirección se halla en la posición $208 y el superior en la $209. Ésta es la conocida convención de direccionamiento *lo-hi* que siguen para su almacenamiento en memoria todas las máquinas 6502. Las direcciones contenidas en cada vector son establecidas por el OS cuando se reinicializa la máquina. ¿Cuál es la justificación de esta alambicada manera de llamar a una rutina del OS? Desde luego no reside en que Acorn esté decidida a hacer la vida del programador lo más complicada posible. Todo lo contrario, el proceso está pensado para facilitársela. ¿Es esto verdad?

Habrá notado que todas las rutinas del OS del BBC mencionadas hasta ahora son llamadas a una dirección contenida en el intervalo $FF00 a $FFFF. No es casual. Una vez llamada esta dirección, se entra en una rutina que ocasiona el salto a la dirección contenida en el vector para esa determinada rutina del OS, como tuvimos ocasión de ver con las llamadas a CLI y a OSCLI. Ahora bien, la dirección que llamamos entre $FF00 y $FFFF es la misma en *todas* las versiones del OS del BBC y así continuará siendo. Si es necesario cambiar los programas internos de la ROM de OS, los diseñadores del OS no tienen más que asegurarse de que las direcciones de las rutinas ROM colocadas en las posiciones del vector han sido alteradas teniendo en cuenta el cambio. Así queda protegido el usuario de tales cambios en el OS siempre que las rutinas del OS sean llamadas en sus puntos correctos de entrada. El contenido, pues, de un vector puede diferir de una versión a otra, pero el usuario no tendrá noticia de ello siempre que use las direcciones del punto de entrada en el intervalo $FF00 a $FFF.

Una segunda ventaja del empleo de vectores es que el método nos brinda un medio de modificar el comportamiento de las rutinas del OS. Basta con alterar el contenido de un vector de modo que apunte a una rutina en código máquina diseñada por nosotros mismos, interceptando así las llamadas normales del OS. Más adelante examinaremos los vectores empleados en las principales rutinas del sistema operativo.

De momento, consideraremos el vector llamado USERV, al que apuntan las posiciones $200 y $201. Se trata de un vector bastante especial, ya que por lo general no hace nada. Se emplea con dos órdenes *, las *CODE y *LINE. Digítelas y se encontrará con el mensaje de error Bad Command. Pero antes de enviar una carta de queja por este nuevo error del OS del BBC lea lo que sigue.

USERV nos permite definir la función realizada con las instrucciones *CODE y *LINE, o sea, instrucciones definidas por el usuario. *CODE es una manera útil de pasar parámetros a programas en código máquina, como se indica en la tabla siguiente:

Registro	*CODE x,y	*LINE cadena texto
A	0	1
X	Contiene el valor del primer parámetro después de *CODE. El parámetro debe estar entre 0 y 255	Contiene el byte inferior de la dir. de memoria donde se puede hallar el primer car. de la cadena después de *LINE
Y	Contiene el valor del segundo parámetro después de *CODE. Otra vez, debe estar entre 0 y 255	Contiene el byte superior de la dir. de memoria donde se encuentra el primer car. de la cadena después de *LINE

Esta tabla muestra el estado de los tres registros de la CPU al entrar en la rutina a la que apunta el contenido de USERV. Se tiene un 0 o bien un 1 en A, para indicar cuál de las dos instrucciones provocó la entrada a USERV. Los valores contenidos en X e Y dependen de la instrucción usada, *CODE o *LINE. Por ejemplo, *CODE 3,2 entrará a la rutina que indique USERV con un 0 en A, un 3 en X y un 2 en Y. Es claro que la rutina señalada por la dirección contenida en USERV será la rutina a la que deseamos pasar los dos parámetros.

El programa que proporcionamos en la página siguiente ilustra un sencillo uso de la instrucción *CODE. La rutina en código máquina se ensambla en la memoria empezando en la dirección $C00 como resultado de la sentencia de asignación de la línea 40: la variable entera P% se asocia al contador de programa del procesador, al igual que hacen A% con A, X% con X e Y% con Y, los registros del procesador. Se activa USERV para que apunte a la rutina colocando el byte inferior de esta dirección, $00, en

instrucciones definidas por el usuario (USERV)

```
10 DIM C 100
20 USERV=&200
30 ?&USERV=&00
40 ?(USERV+1)=&0C
50 FOR I%=0 TO 2
   STEP 2
60 P%=&C00
70 [ OPT I%
80   CMP #0
90   BNE notcode
100  TAX
110  .loop
120  JSR &FFE3
130  DEY
140  CPY #0
150  BNE loop
160  RTS
170  .notcode
180  RTS
190 ]: NEXT I%
200 .
210 FOR rep=1 to 10
220 FOR asc=33 TO 48
230 asc$=STR$(asc)
240 rep$=STR$(rep)
250 code$="*CODE
    "+asc$+","+rep$
260 $C=code$
270 X%=C MOD 256
280 Y%=C DIV 256
290 CALL &FFF7
300 NEXT: NEXT
```

la posición $200, y el byte superior, $0C, en la $201. La rutina visualiza en la pantalla un determinado número de caracteres; el primer parámetro de la instrucción *CODE contiene el código ASCII del carácter y el segundo parámetro corresponde al número de veces que el carácter será visualizado en la pantalla.

*LINE no es de tan amplia utilidad como *CODE, pero si se desea emplearla se pueden aplicar los principios ilustrados en el siguiente programa, siempre que no se olvide entrar al programa con un 1 en el registro A y que los registros X e Y han de apuntar a la cadena de texto en memoria. Ésa es la función principal de *LINE: pasar cadenas de texto a los programas en código máquina. En aquellos casos en que no hay muchos parámetros que pasar a la rutina, estas dos llamadas son el modo más elegante de hacerlo.

La línea 20 del programa activa USERV para que apunte a nuestra rutina en código máquina. El bucle de la línea 90 a la 120 imprime Y veces el carácter cuyo código ASCII contiene el registro A. Si se entra a la rutina por medio de la instrucción *LINE, las líneas 60 y 70 se encargan de detectar el hecho y de abandonar la rutina. Las líneas de la 200 a la 250 generan la instrucción *CODE con los parámetros variables.

Interacción del usuario

Los principales métodos de diálogo con el microordenador son el teclado y la pantalla de televisión o VDU (*Visual Display Unit*: unidad de representación visual). Vamos a seguir investigando en detalle cómo el sistema operativo del BBC nos permite la interacción con estas dos áreas fundamentales del ordenador.

Comencemos examinando la llamada OS que nos permite leer caracteres desde la corriente de entrada seleccionada en un momento dado. Esta rutina, denominada OSRDCH (*OS read character*: lectura de caracteres), es llamada en la dirección $FFE0 y se vectoriza a través de las posiciones $210 y $211. Dado que acepta caracteres uno a uno desde la corriente de entrada seleccionada, vamos a ver cómo se selecciona dicha corriente. Existen dos maneras principales: el teclado y la entrada RS423. Se seleccionan por medio de una llamada OSBYTE o bien *FX. El cuadro siguiente resume esta instrucción en código máquina y en BASIC:

Selección de la corriente de entrada				
n	Teclado	RS423	Ensamblador	BASIC
0	✔	✘	LDA # 2	*FX2,n
1	✘	✔	LDX # n	
2	✔	✔	JSR $FFF4	

Por tanto, *FX2,1 desconecta el teclado y conecta la RS435 como corriente de entrada. Los datos recibidos en la RS423 serán tratados como si hubieran sido digitados. En ensamblador, n debe tener valor 1 si se desea obtener el mismo efecto.

Una vez seleccionada la entrada, se puede acceder a esa corriente por medio de OSRDCH. Lo primero que hay que decir sobre esta rutina es que su utilidad real se demuestra sólo en programas en ensamblador, puesto que el BASIC ya tiene rutinas similares con GET e INPUT. Esta rutina debe ser llamada en la dirección $FFE0 y, a la vuelta de la llamada, el carácter leído de la corriente estará contenido en el registro A; si se detectara cualquier error durante la lectura, el flag de arrastre se pondrá a 1, en caso contrario vendrá a 0. Así, C=1 a la vuelta de la rutina OSRDCH indica que el carácter contenido en A no es válido por cualquier motivo. Cuando la lectura es desde el teclado, el error suele ser debido a la pulsación de la tecla Escape. Esto hace que C se ponga a 1 y el valor contenido en A sea 27 (Escape en ASCII). Si usted se da cuenta de la situación deberá actuar en conformidad y con todo cuidado: el OS del BBC está esperando que el programa le indique que ha recibido este error Escape.

Esto lo haremos empleando una llamada a OSBYTE con A=126, cuyo efecto será el de que se limpien varias partes de la zona de trabajo del OS. Esta operación de respuesta al Escape se hace habitualmente de forma automática por el intérprete BASIC si se pulsa la tecla Escape durante una operación de entrada. La rutina que sigue, muy sencilla, lee la corriente actual de entrada y actúa dependiendo de una posible detección de error de Escape.

```
1000  .input  JSR  &FFE0
1010          BCS error
1020          RTS
1030  .error  CMP#27
1040          BNE out
1050          LDA#126
1060          JSR&FFF4
1070  .out    RTS
```

La línea 1000 llama a la rutina OSRDCH y la 1010 comprueba el estado del flag de arrastre. Si está a cero, entonces se ejecuta una RTS al programa de llamada, con un carácter válido en el registro A. Si no es así, la línea 1030 comprueba si el error fue debido a Escape, y, en caso afirmativo, las líneas 1050 y 1060 ejecutan la llamada a OSBYTE que se encargará de acusar la recepción del evento Escape. Puede que usted piense que para entrar cadenas de datos en sus programas en código máquina deberá emplear una rutina de fabricación propia, pero no es así. Existe en el OS un medio de leer cadenas de caracteres desde la corriente actual de entrada. A esta rutina se accede por medio de una de las llamadas a OSWORD, que habremos de examinar más adelante en este curso. No obstante, usaremos ahora esta llamada particular de OSWORD para leer cadenas de caracteres.

El bloque de control

Las rutinas OSWORD son llamadas en la dirección $FFF1. Existen varias, y para especificar cuál necesitamos nos servimos del valor contenido en A en el momento de hacer la llamada. En todas las llamadas a OSWORD, los registros X e Y del 6502 apuntan a un bloque de memoria llamado *bloque de control*, que contiene los parámetros que se pasan a la rutina. El registro X contiene el byte inferior de la dirección del bloque de control y el registro Y contiene el superior, según la norma seguida por el 6502 del *lo-hi*.

La manera en que se inicializa el bloque de control es la siguiente:

El bloque de control de OSWORD	
Entrada al bloque control	**Función**
0	Byte inferior de la dirección en la que van a ser escritos los caracteres
1	Byte superior de la dirección en la que van a ser escritos los caracteres
2	Longitud máxima de la línea
3	Código ASCII mínimo aceptable
4	Código ASCII máximo aceptable

Durante la introducción de los caracteres por medio de esta rutina, la tecla Delete tiene su función habitual. La rutina puede ser cancelada con sólo pulsar Return o Escape. Por ejemplo, el bloque de control que sigue a continuación tiene estos efectos cuando se hace una llamada a OSWORD:

Ejemplo de bloque de control de OSWORD	
Entrada al bloque de control	**Valor**
0	&00
1	&0C
2	&07
3	32
4	96

1. El primer carácter entrado será almacenado en $C00, el segundo en $C01, etc.
2. Solamente se aceptarán siete caracteres; si intenta digitar algún carácter más, entonces se generará un pitido y todo lo que escriba será ignorado.
3. Solamente se aceptarán códigos ASCII que vayan desde el 32 al 96 (desde el carácter Espacio al carácter £); los demás serán ignorados.

Como puede ver, la llamada nos permite descartar caracteres no deseados a la hora de su entrada. Cuando se ha producido la salida de la rutina, el estado del flag C nos informa de la causa del final de rutina. Si C=1, es que se pulsó Escape. Si C=0, es que Return se encargó de finalizar la entrada de caracteres y el registro Y retiene la longitud de la cadena entrada, incluyendo el valor de retorno de carro en ASCII como final de la cadena. Recuerde que puede servirse de esta rutina en cualquiera de las corrientes de entrada seleccionadas por medio de *FX2 o su equivalente en código máquina.

Acabamos de ver lo fácil que es una lectura de datos en el BBC Micro. Avancemos un poco más examinando los medios de enviar los caracteres a la corriente de salida seleccionada. De nuevo habremos de emplear una llamada del OS para seleccionar la corriente de salida que deseamos usar. Se consigue con *FX3,n, donce n especifica la corriente que se selecciona. Cada bit del parámetro n controla una corriente diferente. Por ejemplo, *FX3,1 permite las salidas seriales, de pantalla o de impresora y permite una salida SPOOL, siempre que haya sido generada una instrucción *SPOOL.

Tabla de parámetros de control corriente salida						
Valor \ **Bit**	**0**	**1**	**2**	**3**	**4**	**6**
1	RS423 ON	Pantalla OFF	Impresora OFF	Impresora ON*	Spool OFF	Impresora OFF**
0	OFF	ON	ON	OFF*	ON	ON**

* Impresora en ON u OFF, esté o no desactivada por otra causa

** Impresora en OFF a menos que el carácter no sea precedido por CHR$(1)

La principal rutina empleada para enviar caracteres a la corriente seleccionada de salida se llama OSWRCH (WR*ite* CH*aracter*: escritura) y es llamada en la dirección $FFEE, vectorizada por medio de las posiciones $20E y $20F. Su empleo es sencillo; basta con cargar el registro A con el código ASCII del carácter que se desea escribir y llamar después la rutina. Las siguientes tres rutinas imprimen el carácter A en pantalla:

```
1000 VDU 65
1000 PRINT CHR$(65)
1000 LDA#65
1010 JSR &FFEE
```

La orden VDU del BASIC produce en esencia los mismos efectos que OSWRCH. Los caracteres del intervalo ASCII 32 a 255 se visualizan en pantalla, salvo el 127, que es el carácter Delete. Los incluidos en el intervalo del 0 al 31 tienen funciones especiales. Ellos son los que nos permiten emplear OSWRCH para dibujar gráficos en pantalla, ejecutar las instrucciones COLOUR y GCOL, definir caracteres y controlar el chip 6845, que es el que controla la visualización vídeo del BBC Micro.

La escritura de caracteres en pantalla o sobre otro soporte cualquiera a través de la rutina OSWRCH es conocida a menudo como escritura sobre *drivers* (manejadores) de VDU. La tabla de códigos de control ASCII ilustra los efectos de los códigos de carácter entre el 0 y el 31 cuando son enviados a *drivers* o unidades VDU. Notará que éstos nos permiten hacer, por medio de rutinas de gráficos en código máquina, todo lo que podemos hacer en BASIC.

Gráficos vía la OSWRCH

Las rutinas OSWRCH nos ofrecen toda la posibilidad de gráficos que obtenemos con instrucciones para gráficos. El programa que servirá de segundo ejemplo, escrito al margen de la página siguiente, nos dibujará en pantalla una línea roja. Las líneas de la 50 a la 75 ejecutan una instrucción GCOL 0,1 estableciendo el color rojo de la línea. Las líneas de la 90 a la 150 ejecutan una instrucción PLOT 5,100,100 que equivale a la orden DRAW 100,100. La línea 100 envía el tipo PLOT (el 5, en este caso) a los *drivers* de VDU, seguido de una abscisa de dos bytes, con el byte inferior primero, y una ordenada de dos bytes, igualmente con el byte inferior primero. Se puede ejecutar una instrucción MOVE sustituyendo el 5 por el 4 (pues MOVE no es más que una instrucción PLOT 4,x,y). Otras operaciones gráficas (como la instrucción PLOT 85 para dibujar triángulos) se pueden obtener por los mismos medios. Pero es im-

Gráficos vía OSWRCH

```
10 MODE 1
20 FOR I%=0TO2STEP2
30 P%=&C00
40 [OPT I%
50 LDA #18
55 JSR &FFEE
60 LDA #0
65 JSR &FFEE
70 LDA #1
75 JSR &FFEE
80
90 LDA #25
95 JSR &FFEE
100 LDA #5
105 JSR &FFEE
110 LDA #0
115 JSR &FFEE
120 LDA #100
125 JSR &FFEE
130 LDA #0
135 JSR &FFEE
140 LDA #100
145 JSR &FFEE
150 RTS
160 ]:NEXT I%
170 CALL &C00
```

portante recordar que, en el envío de instrucciones PLOT a los *drivers* VDU, éstos esperan recibir cinco bytes después de ser enviado el valor 25 como primer byte; si estos bytes no llegan a recibirse, pueden ocurrir efectos extraños. Esto es válido para toda operación sobre *drivers* de VDU que exija el envío de más de un byte.

El VDU23 es otra instrucción útil. Se emplea para definir caracteres generados por el usuario. Por ejemplo, VDU23,224,255,255,255,255,255,255,255, 255 definirá el carácter 224 (por lo general no definido) como un bloque compacto. Los caracteres comprendidos entre los números 244 y 255 en los modos de 0 a 6 pueden redefinirse por el usuario con esta instrucción. De hecho, la instrucción VDU23 en conjunción con una de las llamadas OSBYTE permite al usuario redefinir otros caracteres en el juego de caracteres.

Toda llamada VDU23 no reconocida por el OS (p. ej., VDU23,0...) es pasada por un vector especial situado en $226 y $227. Si se cambia la dirección contenida en este vector, pueden añadirse rutinas propias con la instrucción VDU23.

Un uso más avanzado de la instrucción VDU23 es el de permitir al programador acceder al chip de control de video 6845. Las instrucciones VDU23 toman la forma:

```
VDU23,0,registro,valor,0,0,0,0,0,0
```

aquí registro es el registro 6845 al que deseamos escribir, y valor es el valor que se escribirá en el 6845. Un ejemplo del uso de VDU23 en este sentido es la alteración de dos registros del 6845 en el programa que sigue: dicen al chip cuál es el área de la memoria del ordenador que ha de servir de memoria de video. El programa cambia el inicio de la RAM de video por la dirección $0000. Lo cual muestra la zona de trabajo del OS del BBC sobre la pantalla y pueden verse efectos interesantes. Trate de añadir unas cuantas líneas de código, de dimensionar algunas matrices, etc. La rutina está escrita en BASIC pero se puede convertir fácilmente en ensamblador:

```
10 MODE 0
20 VDU23,0,12,0,0,0,0,0,0,0
30 VDU23,0,13,0,0,0,0,0,0,0
40 VDU28,0,10,30,0:REM establece ventana texto
50 CLS
```

Hay otras dos llamadas del OS relacionadas con OSWRCH. A saber: OSNEWL y OSASCII. La primera, si es llamada en $FFE7, escribe un salto de línea y un retorno de carro en pantalla. OSASCII, llamada en $FFE3, es una variante en OSWRCH, y resulta útil en tratamiento de textos. Cuando se escribe un carácter 13 por medio de esta llamada, el carácter 10 o salto de línea se escribe en la pantalla de salida. No *deberá*, por tanto, ser usada cuando se están escribiendo instrucciones de gráficos a los *drivers* de VDU, ya que puede generarse un CHR$(10) adicional y provocar confusiones.

Finalmente *SPOOL y *EXEC son dos instrucciones que permiten salidas y entradas al sistema de archivos seleccionado. *EXEC nombrearchivo abrirá el archivo cuyo nombre se especifique, si existe, leyendo su contenido como si se tratara de un teclado. *SPOOL escribe caracteres al archivo que se cita en la instrucción, como si tales caracteres fueran escritos a la corriente de salida.

Con lo anterior damos por concluida la introducción al sistema operativo del BBC Micro. En los capítulos venideros nos detendremos en el estudio de las rutinas que mejoran la salida en pantalla del Commodore 64, antes de adentrarnos en una amplia exposición de diversos sistemas operativos.

Tabla cód. de control ASCII

Código	Bytes adicionales requeridos	Descripción
0	0	No hace nada
1	1	Envía el primer carácter a la impresora sólo
2	0	Activa la impresora
3	0	Desactiva la impresora
4	0	Escribe texto al cursor de texto
5	0	Escribe texto al cursor gráfico
6	0	Permite que los drivers de VDU escriban caracteres a la corriente de salida
7	0	Genera un breve sonido
8	0	Mueve el cursor un espacio a la izquierda
9	0	Mueve el cursor un espacio a la derecha
10	0	Mueve el cursor un espacio hacia abajo
11	0	Mueve el cursor un espacio hacia arriba
12	0	Limpia el área de texto de la pantalla
13	0	Devuelve el cursor al inicio de la línea actual
14	0	Activa el modo paginado
15	0	Desactiva el modo paginado
16	0	Limpia el área de gráficos de la pantalla
17	1	Establece el color de texto en el color cuyo código es el byte siguiente
18	2	Hace un GCOL con los dos bytes siguientes que se han de enviar a los drivers. Así, el envío de 18,0,3 ejecutará la orden GCOL 0,3
19	5	Define los colores lógicos. Ver manual
20	0	Devuelve los colores lógicos a los valores por defecto
21	0	No permite que los drivers de VDU escriban caracteres a la corriente de salida
22	1	Pone el modo de pantalla en el modo del byte siguiente. El envío de 22 y 7 dará Modo 7. La HIMEM no se altera
23	9	Envía instrucciones al chip 6845; programa caracteres definidos por el usuario
24	8	Define una ventana de gráficos
25	5	Realiza la instrucción PLOT
26	0	Establece una ventana de texto y gráficos por defecto
27	0	Ningún efecto
28	4	Establece una ventana de texto
29	4	Establece el origen de gráficos
30	0	Lleva el cursor de texto al ángulo superior izquierdo
31	2	Pone el cursor de texto en la posición *x,y* en los dos bytes siguientes. Así, el envío de 31,10,10 pondrá el cursor en la posición 10,10 de la pantalla

Intercambio telefónico

Iniciamos una serie en que analizaremos en detalle las aplicaciones prácticas de las comunicaciones mediante modems

Tono de instrucción
Desarrollado originalmente para conectar terminales remotos a ordenadores centrales, el modem ("*mo*dulador/*dem*odulador") convierte, o modula, datos electrónicos digitales en tonos de audio para transmisiones telefónicas, y demodula los tonos de audio para volver a convertirlos en señales digitales en el momento de su recepción. Los acopladores acústicos transmiten y reciben a través del tubo del teléfono, mientras que los modems "compactos" se conectan directamente a la línea, por lo general a través de un conector de ampliación

La conexión a un modem le permite a su micro "hablar" con otros ordenadores a través del teléfono. De este modo, se abre ante usted una amplia gama de actividades relacionadas con la comunicación: enviar cartas que se reciben en el instante en que se transmiten, intercambiar software con amigos que vivan en lugares alejados, pasarse mensajes con otros usuarios o ganar acceso a un ordenador central. En un futuro artículo analizaremos estas aplicaciones. Aquí vamos a echar una mirada a los principios en los cuales se sustenta la transmisión de datos entre ordenadores.

La tecnología de las comunicaciones (también conocida como *coms*) tiene sus orígenes en la informática de ordenadores centrales. En el pasado se solía situar el ordenador central en una habitación construida expresamente y dotada de aire acondicionado, y desde allí se efectuaban conexiones con terminales distribuidos por todo el edificio. Un terminal constaba tan sólo de una pantalla y un teclado conectados al ordenador principal a través de un cable en serie. De este modo un usuario que estaba sentado en un extremo del edificio podía acceder al ordenador del otro extremo.

El enlace en serie directo entre el ordenador y el terminal funciona correctamente entre distancias relativamente pequeñas, es decir, de unos pocos centenares de metros; pero la pérdida de definición de las señales hace imposible la transmisión a través de distancias mayores, aun cuando el costo del cableado no represente obstáculo alguno. Éste es el motivo por el cual se desarrolló el modem.

Un *modem* (de "*mo*dulador/*dem*odulador") es un dispositivo que permite que los datos del ordenador se transmitan a través de una línea telefónica normal. Funciona convirtiendo las señales eléctricas del ordenador en tonos de audio de frecuencia y volumen aptos para la transmisión a través de la

red telefónica. Este proceso es lo que se conoce como *modulación*. El modem del ordenador receptor vuelve a convertir estos tonos de audio en señales eléctricas que se le puedan pasar al ordenador receptor (lo que se denomina *demodulación*). Se utiliza una señal constante (llamada *tono de portadora*) a modo de referencia, mientras se transmiten los datos en la onda modulada.

El resultado concreto de este proceso es que se puede acceder a ordenadores remotos casi como si estuvieran conectados directamente al terminal. Los terminales exclusivos por lo general están formados por una VDU y un teclado y, puesto que no poseen capacidad de proceso propia, se les suele calificar como "terminales tontos". Se puede emplear un microordenador como si fuera un terminal, simplemente utilizando un software de comunicaciones apropiado. Sin embargo, dado que un micro sí posee capacidad de proceso propia, se dice que en este caso es un "terminal inteligente".

Tipos de modem

Existen dos tipos de modems: acústicos y compactos. Los *modems acústicos* (llamados por lo general *acopladores acústicos*) poseen dos tazas de goma en las cuales se coloca el tubo del teléfono. Los sonidos de audio se transmiten a través de la bocina y se reciben por el auricular. Los modems acústicos son los más convenientes de los dos tipos, porque se los puede utilizar con cualquier teléfono, ¡y los que funcionan a pilas se pueden incluso emplear con ordenadores portátiles para efectuar llamadas desde teléfonos públicos! Por otra parte, los modems acústicos están sujetos a la interferencia del ruido circundante y, por lo tanto, no son muy fiables.

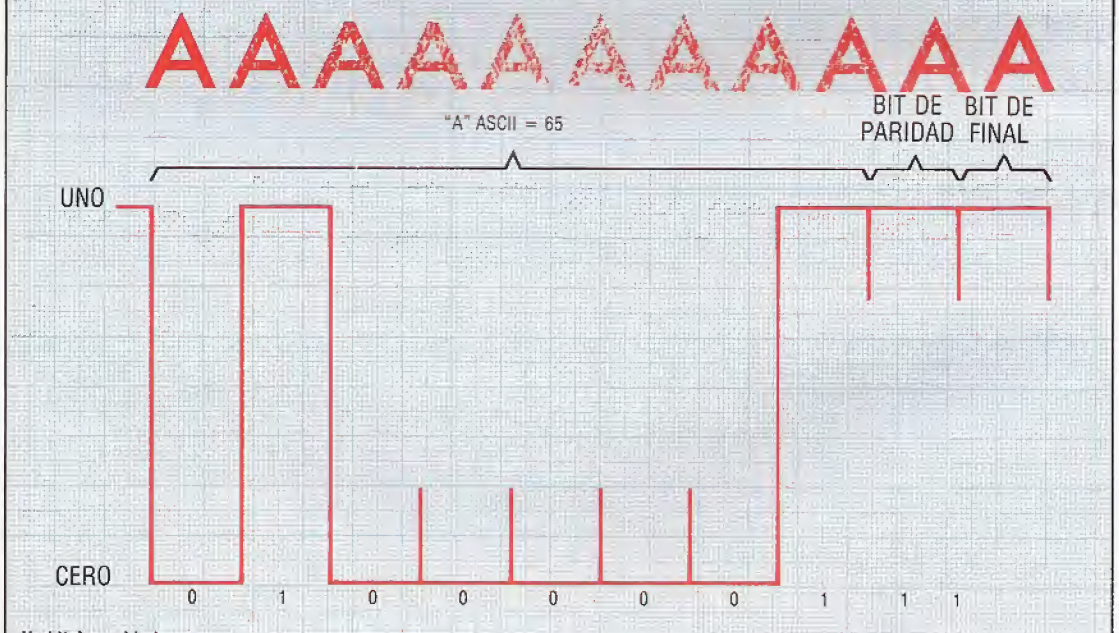
Los *modems compactos* o de "conexión directa" se enchufan directamente en un enchufe estándar de teléfono y el teléfono se conecta en la unidad del modem. Además de ser más fiables que sus equivalentes acústicos, los modems compactos por lo general ofrecen más características.

Debido a que los modems compactos se enchufan directamente en la red telefónica, éstos han de haber sido homologados por la Compañía Telefónica Nacional de España (C.T.N.E.); esto asegura que no exista ninguna posibilidad de que se permita el paso de voltajes de la red eléctrica al sistema telefónico, y verifica que el modem se desconecte "limpiamente" al final de una llamada y no deje la línea "ocupada". El empleo de un modem no homologado es ilegal.

Entre las útiles características adicionales que ofrecen algunos modems se incluyen las de "respuesta automática" y "llamada automática" (esta última, todavía no implementada por la C.T.N.E.). Los modems con respuesta automática, al contestar al teléfono y detectar otro modem al otro extremo del hilo, le pasan el control al ordenador. Si la llamada no es de otro modem, el modem con respuesta automática simplemente cuelga. Los modems con llamada automática pueden aceptar un número desde el ordenador y "marcarlo" automáticamente. Por lo tanto, en respuesta a un nombre que entre el usuario, el software del ordenador puede buscar el número de teléfono en una base de datos y después indicarle al modem que marque ese número.

El *baudio* es la unidad de medida de la velocidad de transmisión de datos entre dos dispositivos. Normalmente se la considera como la cantidad de bits

El tren del pensamiento



Un bit de paridad

Los datos se transmiten por la línea telefónica representando el código ASCII binario de cada carácter como un flujo de tonos audibles. La frecuencia del tono 0 binario está justo por debajo del tono de referencia o "portadora", mientras que el 1 binario posee una frecuencia de tono justo por encima de la misma. Para la detección de errores se pueden generar tonos extras (llamados bits de "final" y de "paridad"): aquí el carácter transmitido es la "A", ASCII 65 o binario 01000001. En este código hay un número par de unos y se está utilizando el sistema de paridad impar (cualquier código contendrá siempre un número impar de unos; el bit de paridad se establece, por lo tanto, en uno). Al mismo le sigue el bit de final, que señala el fin del código del carácter. Este código de carácter de ocho bits requiere 10 bits para su transmisión

que se transmiten por segundo, si bien en la práctica ésta no es una definición exacta, por razones que veremos más adelante en este mismo capítulo.

Las dos velocidades más comunes para los dispositivos de comunicaciones son 300 y 1 200/75 (significando esta última que los datos se reciben a 1 200 baudios y se transmiten a 75). La mayoría de los micros, a título comparativo, guardan los programas en cinta a una velocidad similar (entre 300 y 1 200 baudios).

La velocidad de 300 baudios se utiliza para sistemas en los que se transmite aproximadamente la misma cantidad de datos en ambos sentidos. Éstos incluyen los boletines y los sistemas de correo electrónico como el Telecom Gold. La velocidad de 1 200/75 baudios se emplea para sistemas de videotexto como el Prestel, en los cuales la mayor parte de la información se envía en una sola dirección.

Lamentablemente, tal como ocurre con otros muchos aspectos de la informática, estos estándares no son universales. En razón de las diferencias existentes entre los sistemas telefónicos británico y norteamericano, por ejemplo, los dos países utilizan frecuencias distintas. La frecuencia de Gran Bretaña se conoce como el estándar CCITT (o V21), mientras que la de Estados Unidos es el tono Bell (en honor de la empresa telefónica).

Todo sobre bits

Debido a que el sistema telefónico sólo puede diferenciar con toda fiabilidad una cantidad limitada de frecuencias, los datos se transmiten en forma binaria. Para hacer esto, cada carácter se traduce primero a su equivalente ASCII y luego a su forma binaria. Por consiguiente, la letra "A" se convertiría en 65 (ASCII) y luego en 01000001 (binario). Los unos se representan mediante una frecuencia (justo por encima del tono de portadora y, por ello, se la conoce como "alta"), y los ceros mediante otra (justo por debajo del tono de portadora y, por tanto, considerada "baja").

Dado que el ordenador receptor (conocido como "anfitrión" o *host*) necesita alguna forma de saber dónde termina un carácter y comienza el siguiente, también se utilizan bits de comienzo y de final. Éstos son simplemente frecuencias convenidas: cuando el anfitrión recibe un bit de final, por ejemplo, decodifica los bits precedentes. El protocolo más común para las comunicaciones ASCII es de ocho bits de datos seguidos de un bit de final. Una posible alternativa es la de siete bits de datos y dos bits de final. Otros sistemas utilizan tanto bits de comienzo como de final, indicando el cambio el final de un carácter.

Puesto que la comunicación a través de la red de teléfonos pública no es absolutamente fiable, necesitamos algún método de verificación de errores, gracias al cual el anfitrión pueda asegurarse de que ha recibido un carácter correctamente. La solución más simple es lo que se conoce como "control de paridad". Ésta implica contar la cantidad de bits altos impares (sistema de *paridad impar*) o bien pares (sistema de *paridad par*). Por ejemplo, retomando el ejemplo de la letra "A", el número total de bits altos de 01000001 es dos (un número par). Por consiguiente, de acuerdo a la paridad impar, el bit de paridad también tendrá que estar alto. En el caso de la letra "C", sin embargo, que se traduce como 01000011 en binario, el bit de paridad tendría que estar bajo.

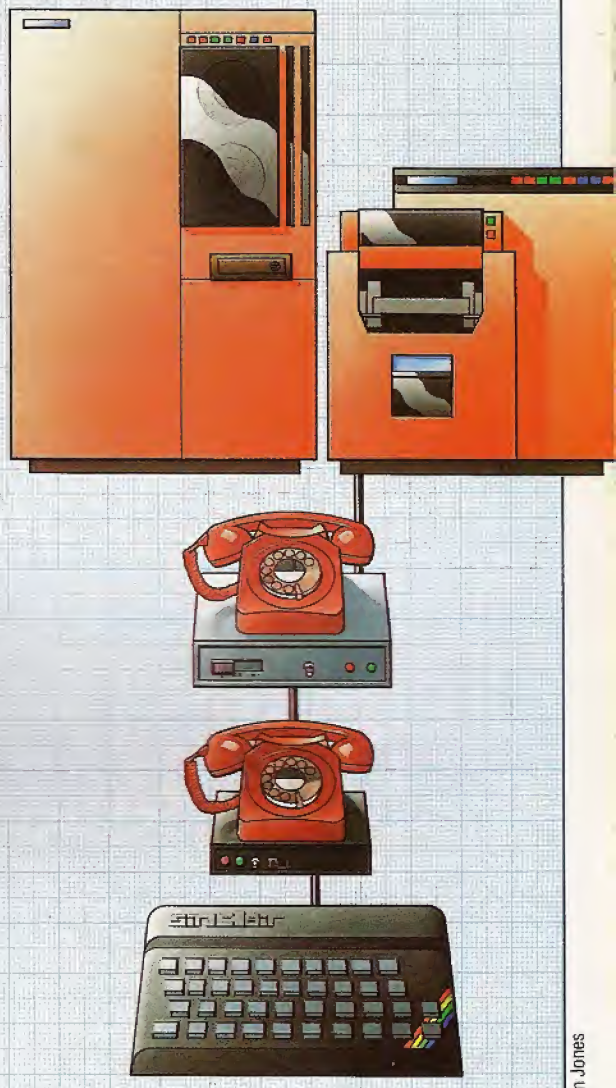
El control de paridad es relativamente poco sofisticado: detectará un error de un único bit, pero no detecta dos errores en el mismo carácter, porque entonces la paridad será correcta. No obstante, es simple y útil para la mayoría de las aplicaciones en las que no resulte vital una exactitud estricta.

Hemos mencionado que la definición de bits por segundo no es exacta para la definición de baudio. El motivo de ello es que se han de tener en cuenta bits de comienzo, de final y de paridad. Por ejemplo, en un sistema con control de paridad que utilice ocho bits de datos y un bit de final para cada carácter, sólo 8 bits de cada 10 transmitidos contienen información útil. Por lo tanto, los verdaderos bits de datos por segundo son el 80 % de 300 baudios, es decir, 240 baudios.

Hemos hecho un análisis detallado de los principios fundamentales de la transmisión de datos mediante modems. En el próximo capítulo consideraremos otros aspectos del funcionamiento del modem, como transmisión dúplex y protocolos de terminales.

Flujo de información

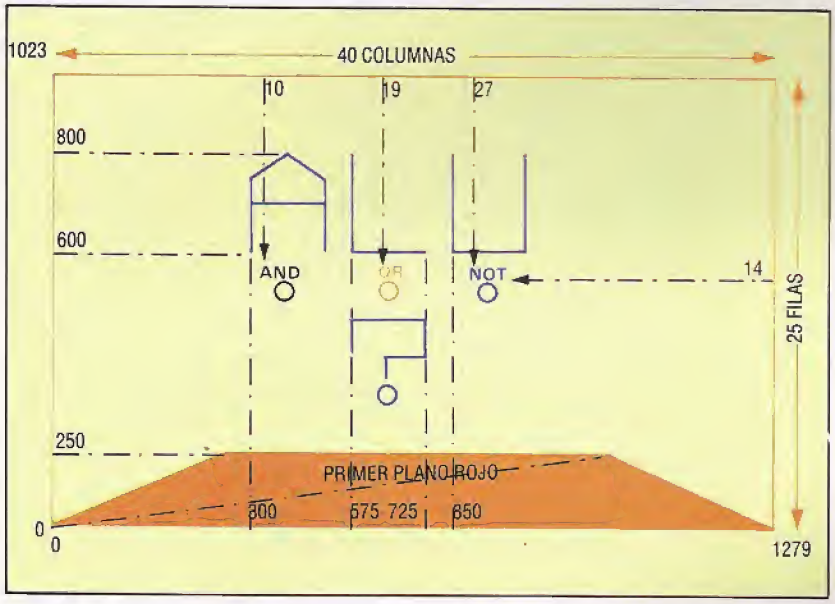
El baudio, así llamado en honor de J. M. E. Baudot, pionero francés de la transmisión de datos, es una unidad de medida de la velocidad del flujo de información entre dispositivos de comunicaciones. Una velocidad de 1 baudio significa que la información se transmite a 1 bit por segundo. Los dispositivos que se comunican a través de un modem suelen utilizar velocidades de 300 baudios cuando se requiere una transmisión aproximadamente equivalente en ambas direcciones. Cuando la transferencia de datos es básicamente en una sola dirección, como sucede en las comunicaciones tipo videotexto, por ejemplo, se emplea una velocidad más rápida, de 1 200 baudios. Estas velocidades de transmisión son muy similares a la velocidad de transmisión de datos empleada para guardar y cargar programas en cinta



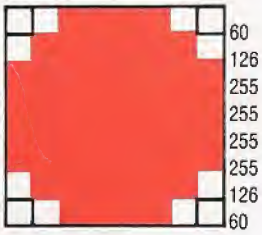
Kevin Jones

Rutinas de pantalla

A lo largo de tres capítulos diseñaremos pantallas de muestra para el BBC Micro, el Commodore 64 y el Spectrum



Mode d'emploi
En un programa para el BBC Micro deben tomarse varias decisiones "comerciales": las modalidades de alta resolución gastan muchísima memoria y soportan pocos colores; las modalidades para texto utilizan menos memoria, permiten mejores gamas de color pero sólo soportan gráficos en resolución media o baja. En este programa, Mode 1 proporciona las resoluciones necesarias, pero al costo de una memoria de pantalla de 20 K



Liz Dixon

El botón
En la imagen de la ALU para el BBC se requiere una forma de botón para representar las tres opciones, AND, OR y NOT. Es preciso redefinir un carácter ya existente. Utilizando una cuadrícula de 8 por 8 podemos diseñar una forma y representarla mediante 8 números decimales. Entonces se puede redefinir CHR$(240) mediante VDU 23:
VDU 23,240,60,126,255,
255,255,255,126,60

En este capítulo veremos cómo se pueden emplear las facilidades para gráficos del BBC Micro para crear visualizaciones en pantalla para juegos de aventuras. El que hemos venido desarrollando, al que hemos llamado *Digitaya*, es un juego de aventuras basado en texto. Es decir, utiliza palabras para describir los ambientes imaginarios en los cuales se sitúa al jugador. Una aventura basada en texto, por ejemplo, simplemente visualizaría el mensaje "Te hallas en la sala del trono" para evocar un escenario, mientras que una aventura gráfica intentaría dibujar una habitación con un trono.

Las pantallas que diseñaremos aquí visualizarán dos escenarios de *Digitaya* que poseen un interés especial: la entrada a la puerta para palanca de mando y la Unidad Aritmético Lógica (ALU). El número de tales pantallas suele estar limitado por la cantidad de memoria disponible; las instrucciones que se requieren para producir cada visualización ocupan un espacio de memoria que, de otro modo, quedaría libre para aumentar la complejidad de la trama argumental.

Diseño de pantalla de la ALU

Antes de que podamos comenzar a diseñar una pantalla para el BBC Micro, debemos responder a varias preguntas:
1) ¿De cuánta memoria dispongo?
2) ¿Cuántos colores necesito?
3) ¿Cuál es la resolución requerida?
Todas estas preguntas en realidad se pueden fundir en una sola: "¿Qué modalidad utilizaré?" Una resolución mayor y una gama de colores más amplia significan que la zona de memoria para pantalla ocupará una valiosa RAM. En nuestro diseño utilizaremos la modalidad 1, que nos ofrece cuatro colores, una pantalla de 40 por 25 y una resolución media. Hemos de establecer la modalidad a utilizar insertando al comienzo del programa la siguiente línea:

```
1095  MODE 1
```

Una vez decidida la modalidad, podemos hacer un boceto de lo que será nuestra pantalla, trazando a lápiz las coordenadas apropiadas a medida que vamos avanzando. El diseño elegido aquí produce el desplazamiento en la pantalla de las letras A, L y U en mayúscula. En el juego, el jugador debe pulsar uno de tres botones (rotulados AND, OR y NOT) y éstos también deben incluirse en la visualización. Características adicionales incluyen un borde estrecho en todo el margen de la pantalla y un primer plano piramidal. A la izquierda de estas líneas podemos apreciar el aspecto gráfico de nuestro diseño primitivo.

Cada letra se forma con un desplazamiento (MOVE) hasta un punto inicial y utilizando luego PLOT 1 para dibujar la forma de la letra como una serie de líneas relativas al punto inicial. Al diseñar las letras de esta forma, las podemos desplazar por la pantalla simplemente cambiando la instrucción MOVE inicial. Asimismo, podemos borrar las letras volviendo a dibujar sus líneas en la misma posición pero especificando un trazado OR-Excluyente mediante el empleo de GCOL 3.

Los botones se forman redefiniendo un carácter. En este caso, CHR$(240) se redefine mediante el procedimiento *botón* para convertirlo en la forma que vemos abajo a la izquierda. Observe que CHR$(240) es asignado a la variable boton$ para utilizar en la parte principal de la rutina. Los botones y las etiquetas se pueden posicionar sólo con visualizarlos (PRINT) en las coordenadas especificadas en la instrucción TAB.

El primer plano se crea empleando las primitivas de relleno triangular que proporciona la instrucción PLOT 85. Esta instrucción une el punto especificado a los dos puntos trazados previamente y luego llena el triángulo resultante con color. La forma cuadrangular del primer plano se puede dibujar y rellenar mediante dos de estas primitivas de relleno.

El código para la visualización en pantalla constituye una subrutina de la rutina especial diseñada para tratar el escenario ALU del juego. La instrucción A$=GET$, de la línea 7560, espera que se pulse una tecla antes de restaurar el color original del primer plano, limpiando la pantalla y retornando a la rutina ALU principal para seguir con el juego. Para llamar a esta subrutina gráfica, también se debe insertar en el programa principal la siguiente línea:

```
4565 GOSUB 7000: REM S/R IMAGEN ALU
```

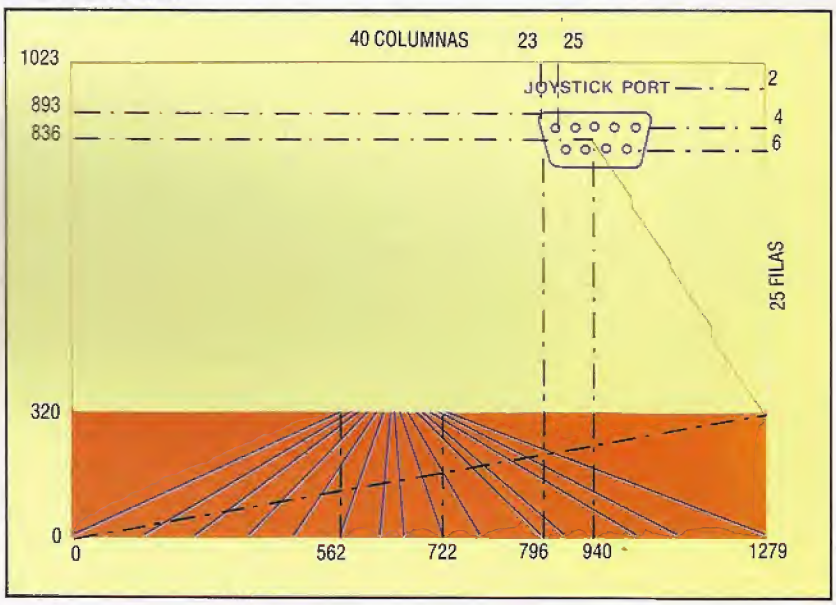
Pantalla de la puerta

En *Digitaya*, si el jugador se extravía en el escenario de la puerta para palanca de mando se encuentra en peligro de ser alcanzado por un rayo láser. El diseño de nuestra visualización en pantalla implica, por consiguiente, dibujar una puerta para palanca de mando de cuyo centro se emitan rayos láser. La puerta para la palanca de mando se dibuja utilizando varios caracteres punto impresos en la esquina superior izquierda de la pantalla, y se dibuja un típico contorno de conector tipo D utilizando gráficos en alta resolución y sentencias PLOT. Observe que después del desplazamiento (MOVE) hasta la posición inicial, todas las sentencias PLOT siguientes que crean la silueta de la puerta son instrucciones PLOT 1, lo que significa que dibujan en relación al último punto trazado. Esto es sumamente conveniente, porque de dibujar las formas empleando una serie de instrucciones relativas, si se decidiera desplazar la posición de la forma completa, sólo debería alterarse la primera sentencia MOVE.

El primer plano está compuesto por un bloque de color rectangular, dibujado empleando nuevamente dos primitivas de relleno triangulares. Para dar la sensación de profundidad, sobre el mismo se dibuja una serie de líneas convergentes utilizando un bucle FOR...NEXT (líneas 8170-8200). El bucle establece el valor de X de 0 a 1280, la anchura de la pantalla en unidades de gráficos. Se trazan hacia la parte inferior de la pantalla una serie de líneas, incrementándose el punto de partida del horizonte para cada punto a medida que se incrementa X. No obstante, en el horizonte el paso de 32, empleado entre líneas consecutivas en la parte inferior de la pantalla, se reduce a un paso de 4 (dividiendo por 8 cada valor de X en la instrucción MOVE que define el punto inicial de cada línea).

El efecto de los rayos láser se produce dibujando una línea que parte del centro de la puerta para la palanca de mando hasta un punto aleatorio del horizonte y con un color al azar. Posteriormente la línea se borra (sin alterar el fondo) trazando la misma línea en la modalidad de trazado con OR excluyente, establecida mediante GCOL 3. El dibujo y borrado de cada línea están ubicados dentro de un bucle REPEAT...UNTIL, junto con una comparación para comprobar si se ha pulsado alguna tecla en el teclado. El empleo de INKEY$ en lugar de GET$ permite que la ejecución del programa continúe mientras dentro del bucle se está comprobando una posible pulsación de tecla. Este bucle termina cuando se pulsa una tecla; entonces se limpia la pantalla, se restaura el color original del texto y se devuelve el control a la rutina principal de la puerta para palanca de mando. Para llamar a esta subrutina de gráficos se ha de insertar la línea siguiente:

```
3845 GOSUB 8000: REM IMAGEN PUERTA PARA PALANCA MANDO
```



Kevin Jones

Sufriendo un rayo
Tanto en la pantalla de la puerta para la palanca de mando como en la de la ALU se hace un uso exhaustivo de la facilidad para trazado relativo, dado que la misma permite un borrado fácil y el movimiento de formas gráficas completas. Para dibujar (DRAW) y rellenar (FILL) bloques sólidos de color en alta resolución se utiliza otra opción de trazado

Pantalla ALU

```
7000 REM ***** SUB PANTALLA ALU *****
7010 CLS
7015 REM ** CURSOR APAGADO **
7020 VDU23,1,0;0;0;0;
7030 REM ** BORDE **
7040 GCOL 0,1
7050 MOVE 0,0
7060 DRAW 0,1023
7070 DRAW 1279,1023
7080 DRAW 1279,0
7090 DRAW 0,0
7095 :
7100 REM ** SENDERO **
7110 MOVE 1279,0
7120 PLOT 85,590,250
7130 PLOT 85,800,250
7140 REM ** LETRA A **
7150 GCOL 3,2
7160 FOR X=0 TO 300 STEP 10
7170 FOR I=1 TO 2
7180 PROCletra_a
7190 NEXT I,A
7200 PROCletra_a
7210 :
7220 REM ** LETRA L **
7230 FOR Y=300 TO 590 STEP 10
7240 FOR I=1 TO 2
7250 PROCletra_l
7260 NEXT I,Y
7270 PROCletra_l
7280 :
7290 REM ** LETRA U **
7300 FOR X=1280 TO 850 STEP -10
7310 FOR I=1 TO 2
7320 PROCletra_u
7330 NEXT I,X
7340 PROCletra_u
7350 :
7360 REM ** BOTONES **
7370 PROCboton
7380 COLOUR3
7390 PRINT TAB(11,15)boton$
7400 COLOUR1
7410 PRINT TAB(20,15)boton$
7420 COLOUR2
7430 PRINT TAB(28,15)boton$
7440 REM ** INSTRUCCIONES **
7450 COLOUR3
7460 PRINT TAB(10,14)"AND"
7470 COLOUR1
7480 PRINT TAB(19,14)"OR"
7490 COLOUR2
7500 PRINT TAB(27,14)"NOT"
7510 :
7520 MOVE 575,400
7530 PROCsigno_l
7540 :
7550 REM ** ESPERAR TECLA **
7560 A$=GET$
7562 REM ** CURSOR ENCENDIDO **
7564 VDU23,1,1;0;0;0;
7566 COLOUR3:CLS
7570 RETURN
7580 :
7590 DEF PROCletra_a
7600 MOVE X,600
7610 PLOT 1,0,150
7620 PLOT 1,75,50
7630 PLOT 1,75,-50
7640 PLOT 1,0,-150
7650 PLOT 0,0,50
7660 PLOT 1,-150,0
7670 ENDPROC
7680 :
7690 DEF PROCletra_l
7700 MOVE 725,Y
7710 PLOT 1,-150,0
7720 PLOT 1,0,200
7730 ENDPROC
7740 :
7750 DEF PROCletra_u
7760 MOVE X,800
7770 PLOT 1,0,-200
7780 PLOT 1,150,0
7790 PLOT 1,0,200
7800 ENDPROC
7810 :
7820 DEF PROCboton
7830 VDU 23,240,60,126,255,255,255,255,126,60
7840 boton$=CHR$(240)
7850 ENDPROC
7860 :
7870 DEF PROCsigno_l
7880 PLOT 1,0,60
7890 PLOT 1,150,0
7900 PLOT 1,0,-70
7910 PLOT 1,-75,0
7920 PLOT 1,0,-50
7930 PLOT 0,-6,-30
7940 PLOT 1,16,0
7950 PLOT 1,0,-16
7960 PLOT 1,-16,0
7970 PLOT 1,0,16
7980 ENDPROC
8000 REM **** IMAGEN PUERTA PARA PALANCA MANDO ****
8010 :
8020 REM ** CURSOR APAGADO **
8030 VDU23,1,0;0;0;0;
8040 CLS
8050 REM ** BORDE **
8060 GCOL 0,1
8070 MOVE 0,0
8080 DRAW 0,1023
8090 DRAW 1279,1023
8100 DRAW 1279,0
8110 DRAW 0,0
8120 :
8130 REM ** HORIZONTE **
8140 PLOT 85,1279,319
8150 PLOT 85,0,319
8160 GCOL0,2
8170 FOR X=0 TO 1280 STEP 32
8180 MOVE 562-X/8,320
8190 DRAW X,0
8200 NEXT X
8210 :
8220 REM ** PALANCA DE MANDO **
8230 COLOUR 2
8240 PRINTTAB(23,2)"JOYSTICK PORT"
8250 PRINTTAB(25,4)" . . . . ."
8260 PRINTTAB(25,6)" . . . . ."
8270 GCOL 0,2
8280 MOVE 796,893
8290 PLOT 1,288,0
8300 PLOT 1,4,-4
8310 PLOT 1,4,-4
8320 PLOT 1,0,-4
8330 PLOT 1,-4,-4
8340 PLOT 1,-30,-81
8350 PLOT 1,-4,-4
8360 PLOT 1,-4,-4
8370 PLOT 1,-216,0
8380 PLOT 1,-4,4
8390 PLOT 1,-4,4
8400 PLOT 1,-30,81
8410 PLOT 1,-4,4
8420 PLOT 1,0,4
8430 PLOT 1,4,4
8440 :
8450 REM ** DISPARO **
8460 REPEAT
8470 A$=INKEY$(10)
8480 X=RND(1279):Y=320
8490 GCOL 3, RND(3)
8500 FOR I=1 TO 2
8510 MOVE 940,836
8520 DRAW X,Y
8530 NEXT I
8540 UNTIL A$<>"":REM ESPERAR PULSACION TECLA
8550 :
8560 REM ** VOLVER A ENCENDER CURSOR **
8570 VDU23,1,1;0;0;0;
8575 COLOUR3:CLS
```

Un buen comienzo

Es esencial que el software educativo para niños tenga un buen diseño y claridad en sus objetivos

Los objetivos didácticos de los paquetes que examinaremos abarcan desde el reconocimiento de colores y formas simples, pasando por aptitudes básicas de lectura y numéricas, hasta refinados intentos para ampliar las capacidades artísticas del niño.

Todos los paquetes son bastante directos al explicarle al usuario lo que tiene que hacer. Ésta debe ser una prioridad en todo programa educativo: el niño debe comprender totalmente lo que se espera de él y se le deben dar recompensas claramente definidas una vez que ha conseguido una habilidad.

En segundo lugar, el programa debe ser fácil de utilizar. No tiene sentido que un programa que pretenda proporcionar al niño aptitudes para la lectura empiece con una lista de instrucciones de funcionamiento. Los mejores programas sólo incluyen un mínimo de estas instrucciones.

Un programa educativo debe suscitar el interés del niño. Independientemente de lo importantes o valiosos que sean sus objetivos últimos, no obtendrá ningún resultado si está por encima de las capacidades del niño o si se vuelve repetitivo y aburrido.

Por último, la prueba de fuego de un programa educativo es que enseñe aquello para lo que fue diseñado. Éste parece un punto obvio, pero con frecuencia las firmas de software olvidan los objetivos didácticos de un programa para favorecer su valor como fuente de entretenimiento.

Los paquetes que analizamos aquí los producen las empresas norteamericanas Spinnaker y Fisher-Price. Aunque estos programas en Estados Unidos existen en formato de cartucho, en Europa se comercializan en cassette. Mientras que a un niño pequeño se le puede mostrar claramente cómo debe insertar un cartucho en el ordenador y encenderlo (para que sea capaz, por consiguiente, de cargar por sí mismo su programa), el formato de cassette exige invariablemente que haya cerca un adulto que lo ayude a cargar el programa.

Dance fantasy

De todos los programas que hemos examinado, tal vez el más atractivo sea *Dance fantasy* (Fantasía de la danza), de Fisher-Price, destinado a niños de entre cuatro y ocho años. La visualización en pantalla muestra un escenario en el que hay dos figuras de pie, y se le solicita al usuario que haga la coreografía de un baile para los dos personajes. Antes de comenzar a trabajar, se pide al usuario que especifique el sexo de los bailarines: un varón y una mujer, o dos varones o dos mujeres.

En la parte inferior de la pantalla el programa visualiza una gama de figuras en diversas poses: cada una representa una rutina de danza particular (un salto, una giga, etc.) Moviendo uno de los bailarines sobre una de estas figuras mediante la palanca de mando el niño efectivamente elige esa determinada rutina, y luego vuelve a situar la figura en el escenario, y accionando el pulsador de disparo obtiene su ejecución. De este modo, se puede realizar la coreografía de un baile seleccionando una serie de movimientos y ejecutándolos en diferentes puntos del escenario, proporcionando el programa los movimientos de conexión. Una vez el niño ha completado un baile, puede guardarlo (SAVE) y contemplar después el efecto global.

Aegean voyage

Como habrá observado a partir de esta descripción de *Dance fantasy*, el punto fuerte del programa consiste en que es una imaginativa analogía de lo que es un programa para ordenador: el niño crea su propio baile (programa) utilizando una serie de rutinas básicas (un conjunto de procedimientos). El mismo se guarda (SAVE) luego en cassette y se carga desde el mismo (LOAD), introduciendo de este modo al niño sin ninguna dificultad en el significado de estos dos términos.

Aegean voyage (Viaje por el Egeo), de Spinnaker, dirigido a un grupo de edades ligeramente mayor, emplea personajes y escenarios de la mitología griega como elementos de un sencillo juego

de aventuras. El objetivo consiste en hacer navegar un barco desde Atenas hasta diversas islas del mar Egeo, evitando rocas y tormentas durante la travesía. Al llegar a la seguridad de un puerto, se visualiza el nombre de la isla y en la parte inferior de la pantalla aparece un mensaje críptico. El jugador debe entonces decidir si explora o no la isla: una decisión correcta le será recompensada con un tesoro, como, por ejemplo, el escudo de Aquiles; una decisión incorrecta determinará el hundimiento del barco a manos de una criatura mítica, como, por ejemplo, Gorgona.

A los escolares entendidos en mitología clásica les resultará desconcertante descubrir que en el juego hay datos erróneos: el Minotauro, por ejemplo, es tan probable que aparezca en la isla de Delos como en Creta. El juego no hace ningún intento por explicar el significado de los nombres o de los lugares: es muy improbable que gracias a esta cassette el niño llegue a adquirir siquiera una educación clásica superficial. También es poco probable que el juego logre mantener el interés del niño durante mucho tiempo, porque los gráficos y el formato carecen de interés y son repetitivos.

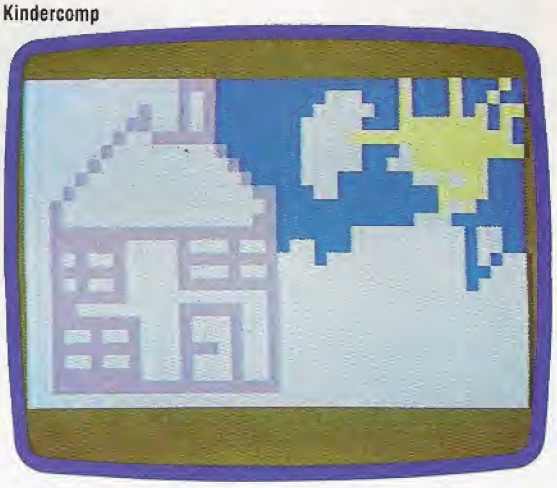
Number tumblers

Diseñado para desarrollar aptitudes para la aritmética mental en niños de entre ocho y doce años, *Number tumblers* (Malabaristas de los números), de Fisher-Price, posee la velocidad y las sensaciones de un juego recreativo. En la parte superior de la pantalla se visualiza una serie de números, y el jugador ha de acomodar los símbolos numéricos y aritméticos de las caras de un conjunto de dados para crear una expresión matemática que sea igual a alguno de los números. El juego es rápido, posee gráficos brillantes y bien diseñados y constituirá un auténtico incentivo para que el jugador perfeccione sus aptitudes de cálculo mental.

Kindercomp, de Spinnaker, está destinado a niños de entre tres y ocho años de edad. El objetivo del paquete es introducirlos en el tema de los ordenadores y desarrollar aptitudes artísticas. El paquete se compone de una serie de ejercicios diferentes.

Este paquete lo escribió originalmente el doctor Doug Davis para su hija, presumiblemente como fuente de entrenamiento. Lamentablemente, *Kindercomp* da la sensación de consistir mayormente en hacer trucos con el ordenador, la clase de cosas que diseñan la mayoría de los programadores cuando están aprendiendo BASIC y descubriendo las capacidades de la máquina. Por ejemplo, una de las opciones es "Nombres". Se invita al usuario a entrar un nombre, o una frase corta, de hasta 15 caracteres de longitud, que se visualiza luego a través de toda la pantalla en diversos tamaños y colores. El efecto es muy atractivo y a un niño que no esté familiarizado con los gráficos por ordenador le parecerá asombroso desde el punto de vista visual. Sin embargo, el programa parece tener muy poco valor educativo, puesto que cualquier grupo de letras proporcionará el mismo efecto.

Kindercomp



Es muy poco probable que *Kindercomp* logre mantener al niño ocupado durante mucho tiempo. Los trucos de programación son divertidos pero enseguida se vuelven repetitivos. Es la clase de paquete que con toda seguridad mantendrá al pequeño usuario ocupado durante tres días más o menos y que después jamás se volverá a utilizar.

El último paquete que examinamos está destinado a los niños de muy corta edad. *Alf in the color caves* (Alf en las cuevas de colores), de Spinnaker, caracteriza a un divertido personajillo que se desliza y se escurre por entre numerosos tubos, llenos de color y de diversas formas, de una habitación de la parte inferior de las cuevas. Utilizando una palanca de mando o el teclado, el niño lo va guiando a través de las cuevas y si lo conduce a salvo sin ser detectado por un par de ojos amenazadores que se mueven muy rápidamente, el niño obtiene como recompensa el espectáculo de ver a Alf ejecutando una deliciosa danza. Luego es absorbido por el tubo hacia arriba, nuevamente hasta el nivel del suelo, para iniciar otro descenso.

Alf in the color caves

Ian McKinnell

Misión espacial

He aquí uno de los programas más interesantes escritos para los ordenadores Atari. Utiliza tanto BASIC como lenguaje máquina

Los movimientos de los diferentes elementos que aparecen en la pantalla están escritos en lenguaje máquina, lo que permite una velocidad superior a la que se obtiene con BASIC. El jugador es el comandante de una nave espacial que se desplaza por una lejana galaxia; dado que la nave ha sido alcanzada por un meteoro, debe, superando cualquier dificultad que se presente, regresar a la base principal para reparar las averías sufridas. ¡Atención!, le quedan sólo 300 l de oxígeno.

Utilice la palanca de mando para conducir su nave. Si tiene éxito en su cometido, la cantidad de aire que quede incrementará su marcador. Quienes se interesen por los subprogramas en lenguaje máquina encontrarán el listado correspondiente a continuación del de BASIC.

```
 10 REM ****************************************
 20 REM *                MISION                  *
 30 REM *               ESPACIAL                 *
 40 REM ****************************************
 50 GOSUB 10000:REM ** INICIALIZACION **
 60 GOSUB 2000
 65 SOUND 0,0,8,15: POKE 53278,R: AIR=300
 70 X=USR(1536)
 80 IF PEEK(781+R*256)=24 THEN 1000
 90 IF PEEK(53254)<>0 OR PEEK(53262)<>0 THEN
    GOTO 2000
100 AIR=AIR-1:IF AIR<>0 THEN 70
110 POSITION 4,6:? "aire agotado":SC=0
115 POKE 53251,0
120 FOR P=0 TO 4000 STEP 12
130 SOUND 0,P,10,15:NEXT P
140 SOUND 0,0,0,0
150 POSITION 4,6:? "pulse START"
155 POSITION 4,4:? "puntos:";SC
160 POKE 53279,0
170 IF PEEK(53279)<>6 THEN 160
180 POSITION 4,6:?                ":GOSUB 10100 GOTO
    65
1000 IF PEEK(209)>PEEK(203) AND PEEK(209)<PEEK(204)
    THEN 1020
1010 GOTO 2000
1020 POSITION 4,6:? "bravo"
1025 POKE 53251,0
1030 FOR P=230 TO 10 STEP -10
1040 FOR X=P-5 TO P+5
1050 SOUND 0,X,10,15
1060 NEXT X
1070 SETCOLOR 1,RND(0)*16,10
1080 NEXT P
1090 SOUND 0,0,0,0
1100 POSITION 21,4:? "BONUS EN AIRE:";AIRE
1110 SC=SC+AIR:GOTO 150
2000 FOR P=781 TO 896:IF PEEK(R*256+P)<>24 THEN
    NEXT P
2005 POKE 53251,0
2010 E=53770:POKE P+256*R+RND(0)+8,PEEK(E)
2020 SOUND 0,PEEK(E),8,15:POKE 706,PEEK(E)
2030 S=S+1:IF S<=50 THEN 2010
2035 POKE 53250,0:POKE 53251,0
2040 S=0:SC=S:GOTO 140
10000 GRAPHICS 0:DL=PEEK(560)+256*PEEK(561)
    +4:POKE 82,0
10010 POKE DL+5,7:POKE DL+6,6:POKE DL+7,6:POKE
    DL+8,6:POKE DL+9,6
10020 SETCOLOR 4,9,4:SETCOLOR 1,3,4:POKE DL+10,6
10030 SETCOLOR 0,3,8:SETCOLOR 1,13,10
10040 POSITION 0,4:POKE 752,1
10050 ? " MISION ESPACIAL ";
10055 ? " ************** ";
10060 ? "estas perdiendo aire ";
10070 ? "y DEBES reunirte ";
10080 ? "con la nave madre ";
10090 ? " ...BUENA SUERTE!!! ";
10100 R=PEEK(106)-8:RESTORE
10110 FOR P=R*256+512 TO R*256+1024
10120 POKE P,0:NEXT P
10130 FOR P=10 TO 17
10140 READ A,B
10150 POKE R*256+512+P,A
10160 POKE R*256+640+P,B
10170 NEXT P
10180 DATA 7,224,62,124,111,246
10190 DATA 255,255,214,107
10200 DATA 124,62,60,60,4,32
10210 FOR P=0 TO 7
10220 READ A
10230 POKE R*256+768+80+P,A
10240 NEXT P
10250 DATA 24,60,36,60,36,126,90,255
10260 FOR P=0 TO 6
10270 READ A
10280 POKE R*256+896+88+P,A
10290 NEXT P
10300 ? CHR$(125):COLOR 160:PLOT 0,20:DRAWTO 39,20:?
    CHR$(160):POSITION 0,0
10310 POKE 704,8*16
10320 POKE 705,8*16
10330 POKE 706,15*16+10
10340 POKE 707,3*16+4
10350 POKE 53277,3
10360 POKE 559,46
10370 POKE 53248,110
10380 POKE 53249,126
10390 POKE 53250,108
10400 POKE 53256,1
10410 POKE 53257,1
10420 POKE 53258,1
10430 POKE 53259,1
10440 POKE 623,1
10450 POKE 54279,R
10460 DATA 24,60,126,255,255,0,0
10470 POKE 203,110:POKE 204,126
10480 POKE 208,INT(RND(1)*2)+1:POKE 209,108
10485 K=R*256+768:POKE 205,INT(K/256):POKE
    205,K-(256*PEEK(206))
10490 RETURN
20000 FOR A=1536 TO 1676:READ B:POKE A,B:NEXT A
20001 DATA 104,185,208,201,1,240,17,230,
    203,230,204,165,204,201,200,208,21,169,1,
    133,208,76,38,6,198
20002 DATA 203,198,204,165,203,201,50,20,8,
    4,169,2,133,208,165,203,141,0,208,165,204,
    141,1,208,173,120
20003 DATA 2,201,11,240,10,173,120,2,201,
    7,240,8,76,72,6,198,209,76,72,6,230,209,165,
    209,141
20004 DATA 2,208,173,132,2,201,0,240,11,
    169,0,141,3,208,141,0,210,76,127,6,173,10,
    210,141,195
20005 DATA 2,165,209,141,3,208,169,130,
    141,0,210,160,0,177,205,136,145,205,200,200,
    192,0,208,245,76
20006 DATA 140,6,160,0,177,205,200,145,205,
    136,136,192,0,208,245,96
20007 RETURN
```

Subprograma

```
10   *=$600
20   ;
30   ;
40   HOR1=$CB
50   HOR2=$CC
60   LF=$D0
70   HOR3=$D1
80   PLA
90   LDA LF
0100 CMP #1
0110 BEQ RIGHT
0120 LEFT INC HOR1
0130 INC HOR2
0140 LDA HOR2
0150 CMP #200
0160 BNE N1
0170 LDA #1
0180 STA LF
0190 JMP N1
0200 RIGHT DEC HOR1
0210 DEC HOR2
0220 LDA HOR1
0230 CMP #50
0240 BNE N1
0250 LDA #2
0260 STA LF
0270 N1 LDA HOR1
0280 STA 53248
0290 LDA HOR2
0300 STA 53249
0310 STICK LDA $278
0320 CMP #11
0330 BEQ L
0340 LDA $278
0350 CMP #7
0360 BEQ R
0370 JMP N2
0380 L DEC HOR3
0390 JMP N2
0400 R INC HOR3
0410 N2 LDA HOR3
0420 STA 53250
0430 LDA $284
0440 CMP #0
0450 BEQ ON
0460 LDA #0
0470 STA 53251
0480 STA $D200
0490 JMP N4
0500 ON LDA 53770
0510 STA 707
0520 LDA HOR3
0530 STA 53251
0540 LDA #130
0550 STA $D200
0560 LDY #0
0570 AG1 LDA ($CD),Y
0580 DEY
0590 STA ($CD),Y
0600 INY
0610 INY
0620 CPY #0
0630 BNE AG1
0640 JMP N5
0650 N4 LDY #0
0660 AG2 LDA ($CD),Y
0670 INY
0680 STA ($CD),Y
0690 DEY
0700 DEY
0710 CPY #0
0720 BNE AG2
0730 N5 RTS
```

Mirada certera

Las más modernas cámaras incorporan microprocesadores, que se encargan de los detalles técnicos necesarios para obtener una buena fotografía

Cuando hace una fotografía, la primera tarea del fotógrafo es decidir la *exposición* correcta. Ello implica determinar cuánta luz de una escena en especial llegará hasta la película de la cámara: si es demasiada, la fotografía quedará descolorida; si es muy escasa, ésta será tan oscura que no se distinguirá nada. Para conseguir la exposición correcta es necesario hallar un equilibrio entre la *apertura* (el tamaño del agujero de la lente, que determina cuánta luz penetra) y la *velocidad de obturación*, que determina la longitud de la exposición.

Por consiguiente, la cantidad de luz de una escena en particular se debe primero medir y, teniendo en cuenta la sensibilidad de la película, determinar en consecuencia la apertura y la velocidad de obturación. Con el correr de los años se han desarrollado medidores que permiten al fotógrafo efectuar una medición fiable del brillo de una escena. Más recientemente se han incorporado a las cámaras medidores de exposición, si bien el fotógrafo todavía debe seleccionar una velocidad de obturación y la apertura de acuerdo a la lectura del medidor.

Los avances que experimentó la electrónica en los años setenta ha hecho posible traducir la lectura del medidor de luz directamente en puntos para la apertura o la obturación. Ello se realiza sin que el fotógrafo intervenga en absoluto, de modo que se pueden obtener resultados de buena calidad simplemente dirigiendo la cámara hacia un determinado punto y disparando. Esta facilidad es particularmente útil tanto para el principiante, que desea tomar fotografías sin entender cómo funciona la máquina, como para los fotógrafos profesionales, que muchas veces necesitan hacer fotografías en condiciones inadecuadas.

La Canon A1 ofrece seis modalidades diferentes para hacer fotografías. Éstas son:

1) *Prioridad a la obturación:* El usuario selecciona una velocidad de obturación y la cámara establece la apertura correspondiente.
2) *Prioridad a la apertura:* El usuario selecciona una apertura y la cámara establece la velocidad de obturación.
3) *Programa:* La cámara establece tanto la velocidad de obturación como la apertura mediante un programa que las combina óptimamente.
4) *Flash automático:* Si está equipada con ciertos flashes, la cámara establece de forma automática la velocidad de obturación adecuada para flash (1/60 segundo) y establece la apertura correspondiente para éste. Un medidor de luz situado en el flash reduce el poder de éste cuando ya ha rebotado suficiente luz desde el tema.

Cámaras eficientes
Desde los años setenta los microprocesadores han venido controlando las funciones interrelacionadas de la medición de luz y el punto de apertura. Las cámaras controladas por microprocesador de hoy en día son capaces de mucho más: se ocupan de la obturación, la exposición y las funciones del flash para permitir que un usuario novato realice tomas de gran calidad, aun en condiciones de luz desfavorables. Dos de tales cámaras son la Nikon FA y la Pentax Super A, capaces de efectuar programas alternativos de control que abarcan una gama de situaciones diferentes

Ian McKinnell

5) *Prioridad de apertura diafragmada:* Es para lentes de tipo antiguo y ciertos accesorios que funcionan con la apertura de la lente siempre "diafragmada" al valor utilizado para tomar la fotografía. Las lentes modernas permanecen en el punto máximo de apertura, excepto en el instante en que se toma la fotografía, para mantener la imagen del visor lo más brillante posible.
6) *Manual:* El fotógrafo establece tanto la velocidad de obturación como de apertura. Es útil para obtener un control total cuando se desea un efecto especial o la iluminación es extraña.

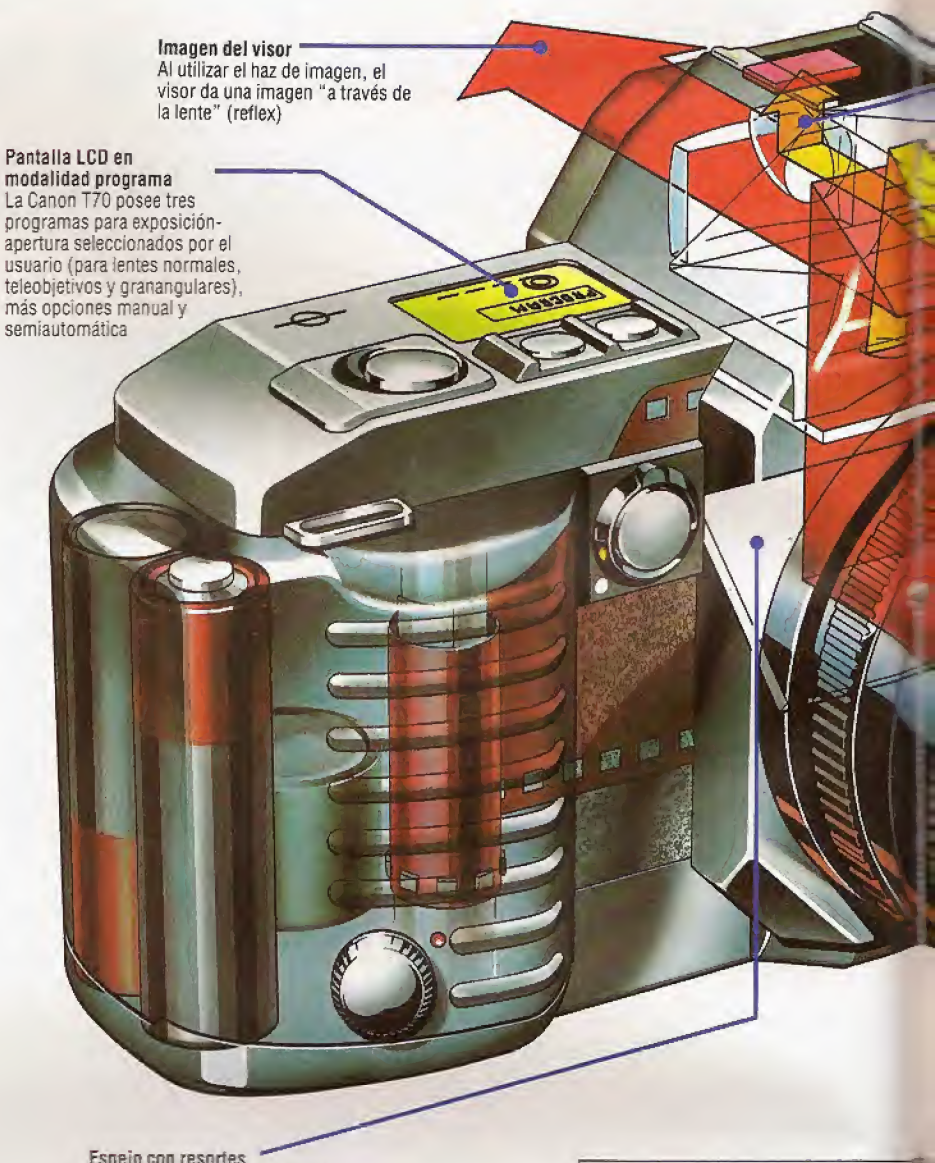
Una visualización electrónica dentro del visor de la Canon A1 indica al fotógrafo en qué modalidad está la cámara y qué valores ha establecido para la obturación y la apertura. La visualización emplea LED para que sea visible incluso en la oscuridad.

A pesar de la utilidad general de la modalidad programa de la Canon A1, ésta funciona de acuerdo a una fórmula bastante simple. Esto significa que, en unos pocos casos, no selecciona la mejor combinación posible. Por ejemplo, si se estuvieran sacando fotografías a última hora de la tarde, la cámara podría seleccionar una velocidad de obturación de 1/30 segundo y una apertura de f/2.8. Las fotografías tomadas a una velocidad de obturación inferior a 1/60 corren el riesgo de arruinarse por los imperceptibles movimientos de la mano del fotógrafo (lo que se conoce como "temblor de la cámara"). Cuando la velocidad de obturación desciende de 1/60 segundo, la cámara hace centellear un aviso previniendo del peligro del temblor de la cámara, pero el programa no selecciona una apertura mayor para permitir la mayor velocidad de obturación.

La competencia de Canon

Algunas empresas rivales han lanzado cámaras de modalidades múltiples con microprocesadores incorporados. La Pentax Super A utiliza un programa ligeramente más sofisticado que la Canon A1. que le proporciona una mejor combinación de velocidades de obturación y aperturas tanto con luz intensa como con luz tenue. En la situación de "últimas horas de la tarde" que describíamos antes, la Pentax Super A seleccionaría una velocidad de poco menos de 1/60 segundo, de modo que habría menos posibilidades de que se produjera un temblor de cámara. Y la Nikon FA, fabricada por el rival por excelencia de Canon, detecta automáticamente cuándo hay instalado un teleobjetivo (longitud focal de 135 mm o más) y accede a un programa alternativo. Éste está optimizado para evitar el temblor de cámara con una lente mayor mediante el empleo de velocidades de obturación más rápidas y mayores aperturas.

Haciéndose eco de la tendencia de Nikon, Canon ha utilizado tres programas alternativos en su última cámara, la Canon T70. Uno es para lentes normales, otro para teleobjetivos y el tercero para lentes granangulares. No obstante, la cámara no reconoce automáticamente cuál es la lente instalada, por lo que el usuario debe seleccionar el programa apropiado. Esto no representa necesariamente un inconveniente, puesto que permite un pequeño control creativo adicional. Por ejemplo, si usted está disparando hacia un punto que se mueve rápidamente con una lente granangular, puede seleccionar la modalidad de teleobjetivo para obtener grandes velocidades de obturación y asegurar que la acción se congele.

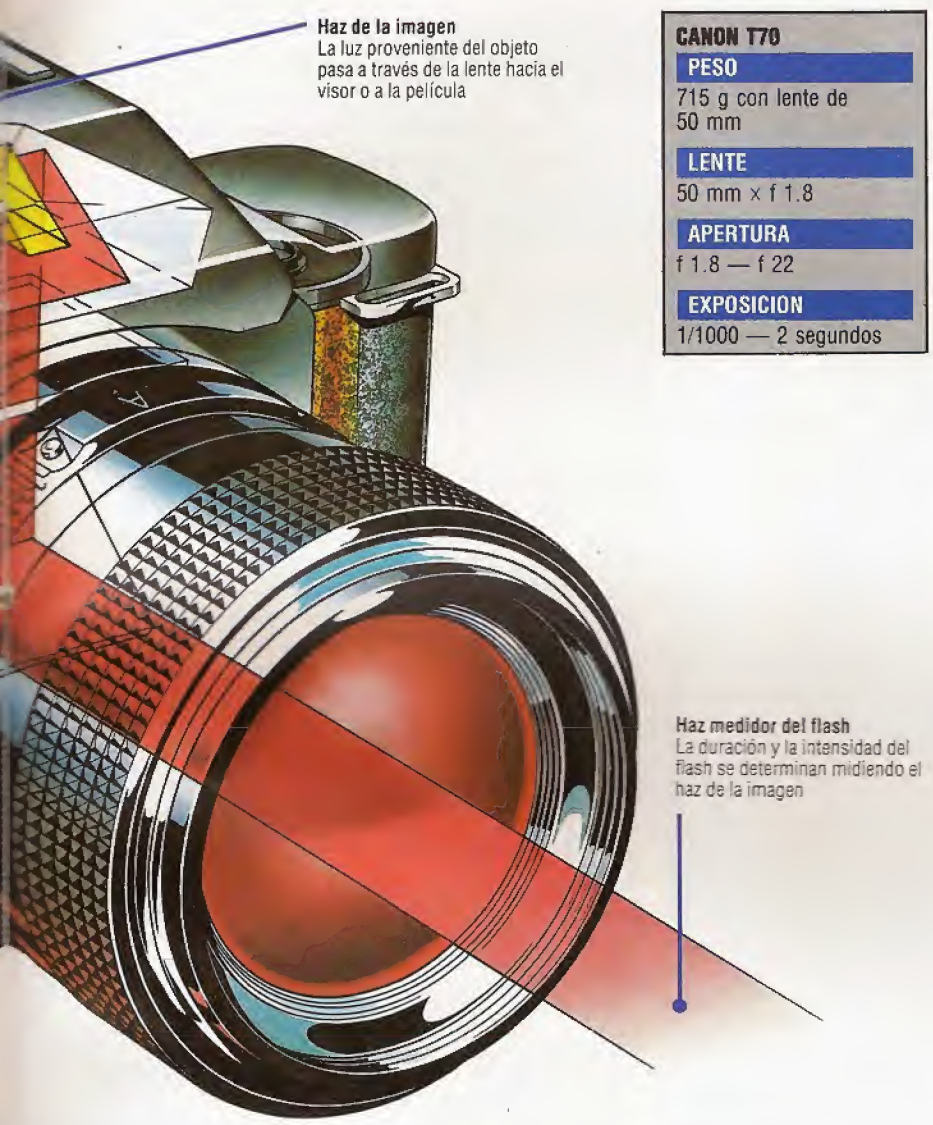


Otro problema de las cámaras automáticas es que dan una exposición promedio para toda la imagen, y los objetos con una extremada gama de brillo en la imagen pueden engañar fácilmente al medidor. Por ejemplo, si se fotografía una motocicleta contra un fondo de atardecer, la cámara tenderá a dar la exposición correcta para el sol y hará que la moto quede demasiado oscura. Por otra parte, si la motocicleta se fotografiara contra un fondo oscuro, la cámara trataría la escena como si fuera mucho más oscuro de lo que en realidad es y probablemente la fotografía quedaría sobreexpuesta.

La Nikon FA utiliza una novedosa forma de abordar el problema. En vez de tomar una sola medida del brillo de una escena, mide cinco partes diferentes de la misma. La FA emplea luego un microprocesador para comparar las cinco lecturas con varias "escenas estándares" programadas en la cá-

Microfotos

Una CPU de ocho bits construida especialmente controla la operación global de la Canon T70, con la ayuda de chips de medición y de ISO. El oscilador temporizador de cristal genera los impulsos síncronos del reloj y controla la longitud de la exposición. Unos contactos controlados por el medidor IC establecen la apertura. Se puede incorporar un módulo opcional de instrucciones que ofrece exposición automática a intervalos (entre un segundo y un día) y permite escribir directamente en el negativo los datos de temporización



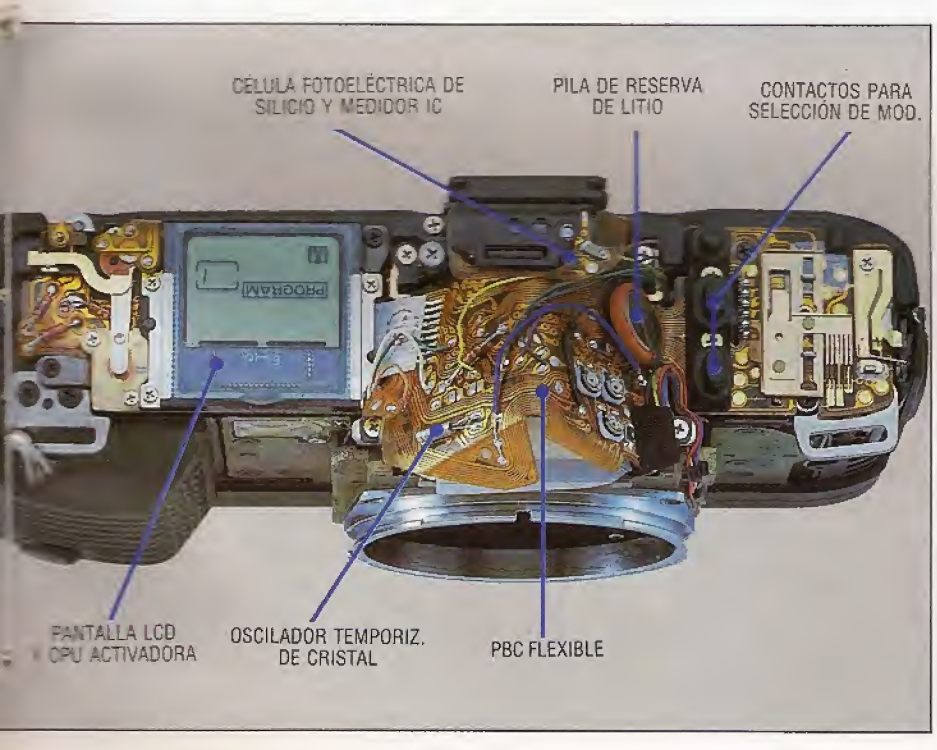
CANON T70

PESO
715 g con lente de 50 mm

LENTE
50 mm x f 1.8

APERTURA
f 1.8 — f 22

EXPOSICION
1/1000 — 2 segundos



mara. Cada una de estas escenas se produce analizando centenares de fotografías.

Pero de todas las cámaras disponibles en la actualidad, la que hace el uso más cabal de la electrónica es la Canon T70. La T70 no posee controles mecánicos; todos sus ajustes se efectúan accionando pulsadores. Se puede seleccionar una de sus ocho modalidades accionando un pulsador situado en la parte superior izquierda de la cámara. Éste hace que aparezca información en una gran LCD situada en la parte superior derecha de la cámara. La información importante, como la velocidad de obturación, la apertura y la modalidad, también aparece en el visor, de manera que el fotógrafo al encuadrar una fotografía no tiene que alejar la cámara de su ojo.

La Canon T70 en acción

Cuando se selecciona una velocidad de obturación, los ajustes se efectúan en relación al valor actual visualizado en la LCD, mediante dos pulsadores que aumentan o disminuyen la velocidad. La sensibilidad de la película (conocida como su número de ASA o ISO) se establece de forma muy similar. El contador de disparos, que muestra cuántas fotografías se han tomado, también aparece en la visualización en cristal líquido.

La máquina posee un motor incorporado para avanzar la película y rebobinarla cuando se termina el carrete. La T70 funciona con dos pilas normales, y en la LCD hay tres barras que indican su estado. Si aparecen las tres barras, las pilas están nuevas; dos barras señalan que están parcialmente agotadas, y una barra avisa que es necesario sustituirlas. Si se utiliza el disparador automático, una visualización en la LCD cuenta hacia atrás los segundos hasta liberar el obturador.

Los microprocesadores que se utilizan en las cámaras son mucho menos potentes que los que se emplean en los ordenadores. La T70 posee su propio microprocesador de ocho bits de fabricación especial, que es de tipo CMOS para mantener al mínimo su consumo de energía. Opera a una velocidad de reloj de sólo 32 KHz; los microordenadores trabajan aproximadamente 100 veces más rápido. Cuando no se la está utilizando, la cámara pasa a unos magros 8 KHz para economizar potencia.

El microprocesador está alojado en un paquete plano de 60 patillas y posee una gran cantidad de ROM pero sólo 16 K de RAM. Con el microprocesador trabajan otros cuatro chips, siendo el más importante el de entrada/salida. Éste controla la operación mecánica de la cámara gracias a unos imanes y un motor. Asimismo, convierte la señal eléctrica analógica del chip del medidor de luz en una señal digital que pueda ser comprendida por el microprocesador.

El poder de la microelectrónica hace posible disfrutar enormemente captando fotografías de gran calidad sin tener que comprender las complejidades de la fotografía. Aun así, siempre habrá casos en los cuales la cámara dará una exposición incorrecta o enfocará el objeto equivocado. La persona que realmente entienda el funcionamiento de una cámara siempre estará en una situación ventajosa respecto a un principiante que posea un equipo fotográfico sofisticado, pero año tras año esta ventaja se está reduciendo cada vez más.

Fuente de energía

En esta ocasión analizaremos los servomotores y sugeriremos algunos usos a los que se pueden destinar

Existen tres tipos de motores eléctricos: de corriente directa, paso a paso y servo. Un motor de *corriente directa* (CD) se puede controlar fácilmente por ordenador, pero tiende a ser impreciso si encuentra cualquier resistencia. En tal situación se reduce la velocidad del motor y el ordenador no puede mantenerse informado sobre su posición.

Un motor *paso a paso* no tiene este problema, porque se mueve en intervalos de ángulo fijo (por ejemplo, 7,5°) cada vez que se le da un único impulso de corriente. Contando los impulsos, y suponiendo que el motor no esté nunca sobrecargado, el ordenador puede calcular la posición del motor. Los motores paso a paso se utilizan mucho en los sistemas controlados por ordenador, como brazos-robot, tornos, distribuidores, etc. Sin embargo, los impulsos de control también le distribuyen energía al sistema y, por lo tanto, los motores requieren activadores construidos especialmente.

En las tiendas de maquetas y modelos a escala se pueden adquirir pequeños *servomotores* digitales, ya que éstos se utilizan comúnmente para los aviones, barcos, coches, etc., controlados por radio. El tamaño de estos motores varía desde aproximadamente la mitad de una caja de fósforos hasta casi diez veces ese tamaño. Algunos servomotores son sumamente fuertes y capaces de proporcionar más "momento de torsión" (término que se utiliza para describir cualquier fuerza que provoque una rotación) del que podrían producir la mayoría de las personas empleando un destornillador grande. Incluso los motores más baratos de este tipo son adecuados para hacer brazos-robot pequeños, etc.

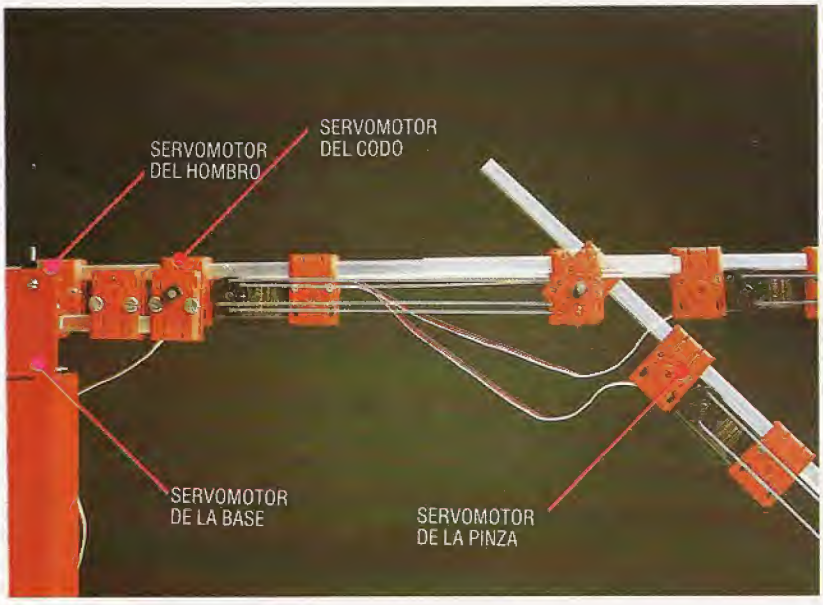
Un servomotor digital pequeño, como un Futuba FP-S126 o un Axoms AS-1, contiene un potenciómetro de realimentación y un diminuto motor CD enlazado (mediante una serie de engranajes) a un "asta". Esta última es una prominencia del motor en la cual se pueden fijar palancas, dientes de engranaje, etc. El motor también es ideal para montar un sistema de bucle de realimentación con un ordenador, ya que la caja del motor también contiene todo el sistema de circuitos necesarios para hacerlo, así como un chip controlador del circuito integrado.

Un servomotor digital típico de modelista se alimenta desde una fuente de 5 V y su ángulo (posición) se establece a través de un cable de control separado.Un impulso de un milisegundo (1/1000) desplazará el asta en una dirección, mientras que un impulso de dos milisegundos la moverá en la otra dirección, describiendo el mismo ángulo. Las variaciones en el ángulo de movimiento son proporcionales a la duración del impulso.

Sin embargo, el motor permanecerá activo sólo durante unos 20 milisegundos después del impulso, momento en que se "relaja" y retorna a su posición original. Por consiguiente, para mantener la palanca en un ángulo determinado, el impulso de control debe repetirse a una frecuencia de unos 50 Hz.

Los servomotores se suelen emplear para mover palancas, etc., pero también pueden ser utilizados para el movimiento lineal. Si el potenciómetro se desacopla del tren de engranajes y se centra, el motor girará continuamente. De hecho, la velocidad a la cual gira está determinada por la duración del impulso.

Bestia de carga
El brazo-robot Beasty está alimentado por tres servomotores (giros de base, hombro y codo) con un cuarto servomotor opcional para activar el efector final. El servomotor es ideal para el brazo-robot, precisamente porque se lo puede mover y después fijar en esa posición



Ian McKinnell

Conexión al ordenador

Cuando los servomotores se conectan directamente a la puerta para el usuario de un ordenador, los errores en el cableado pueden dañar los delicados mecanismos internos de la máquina. Por lo tanto, se ha de utilizar un sistema de tamponamiento, y las cajas buffer y de salida que construimos anteriormente en este apartado son ideales para este fin. Usted debe conectar el cable de la fuente de alimentación eléctrica del motor al conector positivo (rojo) de una de las líneas de la caja de salida de bajo voltaje. El cable a tierra común se debe conectar al conector negativo (negro).

Si el cable de control del motor se ha conectado a la línea de datos 0 de la puerta para el usuario del ordenador, entonces se puede controlar el motor propiamente dicho enviando a la línea de datos 0 de la puerta para el usuario las secuencias de impulsos apropiadas. Un impulso se envía elevando la línea 0 a 5 V, mediante el almacenamiento de un 1 binario en la línea 0. Se utiliza entonces un bucle de cuenta hacia atrás para esperar el tiempo deseado antes de volver a reducir la salida mediante el almacenamiento de un 0 binario en la línea 0.

Tanto el BBC Micro como el Commodore 64 utilizan chips adaptadores para interface versátiles para conformar la puerta para el usuario. Dado que una puerta para el usuario se puede emplear tanto para entrada como para salida, la que estamos utilizando se debe establecer primero en la modalidad requerida (en este caso, para salida). En ambos micros esto se puede realizar manipulando (POKE) directamente el registro de control de dirección de datos usando BASIC.

Ahora debemos considerar cómo enviar impulsos por una línea de datos de la puerta para el usuario. Si se colocara (POKE) en ésta un valor de 88 hexadecimal (equivalente al 136 decimal, o a un patrón de bits binario de 10001000), los voltajes de las patillas de datos serían: 5 V, 0 V, 0 V, 0 V, 5 V, 0 V, 0 V, 0 V respectivamente. Este patrón permanecería constante hasta que fuera modificado deliberadamente. Por consiguiente, se puede generar un impulso simple en la línea de datos 0 colocando (POKE) hexa 00, hexa 88, hexa 00.

Para que el impulso sea lo bastante rápido, se debe utilizar código máquina. El algoritmo para enviar impulsos a un único servomotor es:

1) Especificar el ángulo de la palanca almacenándolo en un solo byte (llamado ANGULO) con un valor entre 0 y 255, utilizando un programa en BASIC.
2) Establecer alta (5 V) la línea de datos 0, comenzando de este modo el impulso.
3) Esperar un milisegundo mediante bucle y decremento del contador.
4) Esperar durante otro período de entre 0 y 1 milisegundo, nuevamente mediante bucle, pero esta vez el valor de partida del contador (y, por tanto, la cantidad de iteraciones) es ANGULO.
5) Poner baja (0 V) la línea de datos 0 para terminar el impulso.

Si ANGULO=0, el impulso durará un período de un milisegundo; si ANGULO=128 (en el BBC Micro y el Commodore 64) durará 1,5 milisegundos y la palanca se moverá hasta un punto medio.

Un impulso, no obstante, no es suficiente para mantener la posición de un servomotor. Éste debe recibir una corriente de impulsos, refrescando al motor aproximadamente cada 20 milisegundos. Existen dos maneras de generar una corriente de impulsos:
1) Simplemente empleando un bucle de espera para hacer una pausa entre los impulsos. Esto significa que mientras se efectúa el bucle el ordenador no puede hacer ninguna otra cosa.
2) Utilizando "interrupciones", que permiten que el ordenador ejecute otro programa (por lo general en BASIC) casi simultáneamente. Este programa de segundo término puede indicar a los motores hacia dónde moverse.

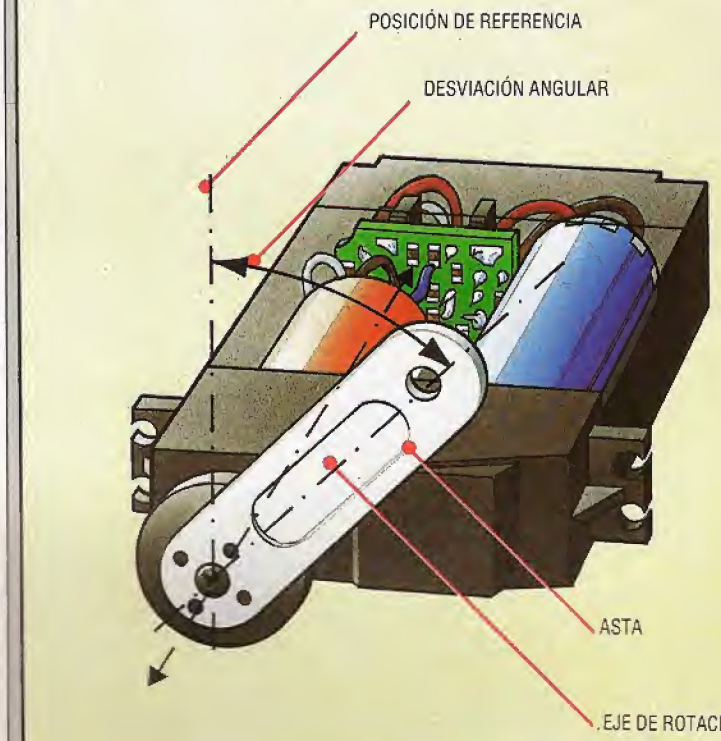
Tanto el BBC Micro como el Commodore 64 utilizan procesadores de la serie 6500, que poseen dos patillas para interrupciones: NMI e IRQ. La segunda, la línea de interrupción, la emplearemos para nuestras tareas de temporización. Cada vez que en la línea IRQ aparece un impulso, el procesador interrumpe lo que esté haciendo y comienza a ejecutar el programa de tratamiento de interrupciones. Una vez completado, retorna (RTI) al mismo punto en que se encontraba cuando fue interrumpido.

Tanto el BBC Micro como el Commodore 64 utilizan un sistema de interrupciones para sus sistemas operativos. El BBC Micro genera 100 interrupciones por segundo (una cada 10 milisegundos) y el 64 posee un coeficiente de 60 por segundo. A cada interrupción se actualizan los temporizadores del sistema, se explora el teclado, etc. De este modo, los sistemas operativos de estas máquinas poseen relojes que generan interrupciones y poseen, asimismo, manejadores (*handlers*) para interceptarlas y usarlas.

En los dos ordenadores, las interrupciones del sistema se pueden utilizar para ejecutar el programa generador de impulsos. Las interrupciones del Commodore 64 deben interceptarse cambiando el vector de interrupción. Este vector (una dirección de dos bytes retenida en dos celdas consecutivas) le dice al procesador la posición de la rutina que trata las interrupciones. Cambiando esta dirección al punto de la rutina de impulsos, y dirigiendo al procesador otra vez hasta el manejador de interrupciones del sistema, al final del impulso, el procesador generará un impulso cada vez que se produzca una interrupción: es decir, 60 veces por segundo.

Aquí le ofrecemos las versiones en BASIC y assembly de un programa para controlar un solo servomotor en el Commodore 64.

También sirven...



Kevin Jones

La desviación del asta fijada al husillo del servomotor está determinada por la duración del impulso enviado por el ordenador controlador; cuanto más largo es el impulso, mayor es la desviación. Si el impulso se repite a intervalos de unos 20 milisegundos, el motor mantendrá el asta en una determinada posición: este "bloqueo de posición" hace que el servomotor resulte ideal para aplicaciones de control electromecánico. La mayoría de los microordenadores generan interrupciones cada 10 o 20 milisegundos, de modo que esta señal de "refresco" se puede enviar sin referencia a un programa de control en BASIC, parcheando algo de código máquina en el sistema operativo a través del vector de interrupción.
Otra ventaja de los servomotores es que sólo se necesita una línea de datos para enviar los impulsos de control y refresco, de modo que a cada servomotor se le otorga un solo bit de la puerta de datos del ordenador controlador

Control de un solo servomotor mediante el Commodore 64

La primera parte del listado fuente para el control de un solo servomotor mediante el Commodore 64 muestra cómo se alteran los vectores de interrupción (posiciones 788 y 789). Esto no se puede hacer desde BASIC, puesto que durante esta alteración se podría producir una interrupción, lo que haría que el sistema se colgara. Observe que las interrupciones se inhiben (SEI) mientras se produce la alteración y se reactivan utilizando CLI. El resto del código es la rutina para tratamiento de interrupciones para controlar un servomotor.
El programa de llamada en BASIC muestra todo lo necesario para cargar las rutinas en código máquina, prepara la puerta para el usuario y después coloca (POKE) valores en la posición de memoria $3000 (12288) según la tecla (de 1 a 9) que se pulse. Un motor conectado a la puerta para el usuario debería entonces de moverse hasta una posición proporcional al valor de la tecla. Pulsando E se da por concluida la sesión. Si usted posee un ensamblador, entre el listado fuente y ensámblelo en un archivo objeto que subsiguientemente se pueda cargar mediante el programa de llamada en BASIC. De no ser así, entre el cargador en BASIC para el código máquina y ejecútelo para cargar el código en la memoria. Digite NEW antes de cargar y ejecutar el programa. Si utiliza el cargador en BASIC, puede omitir las líneas 30 y 40.
Nota: Si hay algo mal en un programa que utiliza interrupciones, es muy fácil que todo el sistema se corrompa. Por tanto, conviene guardar el programa antes de ejecutarlo

Código fuente

```
;.....
;.....
;..                    ..
;..    MANEJADOR DE UN  ..
;.. ..... SERVO CBM     ..
;.....
;.....
;
PUERTA=56579   ; REG DATOS PUERTA USUARIO
ANGULO=12288   ; POSICION VALOR ANGULO
;
*=$0334
;
        SEI            ;APAGAR INTERRUPCIONES
        LDA $0314      ;VECTOR IRQ EXISTENTE
        LDX $03C4
        STA $03C4
        STA $0314
        LDA $0315
        LDX $03C5
        STA $03C5
        STX $0315
;
        CLI            ;VOLVER A ENCENDER
        RTS            INTERRUPCIONES
;
;... MANEJADOR DE EVENTOS ...
;
        PHP
        PHA
        TYA            ;SALVAR REGISTROS
        PHA            ;EN LA PILA
        TXA
        PHA
        LDA #$FF
        STA PUERTA
        LDY #$FF
BUCLE
        DEY            ;BUCLE DEMORA
        BNE BUCLE      ;APROX 1MSEG
;
        LDY ANGULO
BUCLE1
        DEY            ;DESCUENTA IMPULSO
        BNE BUCLE1
;
        LDA #$00
        STA PUERTA     ;REGISTRO DATOS CERO
;
        PLA
        TAX            ;RESTAURAR REGISTRO
        PLA            ;VALORES
        TAY
        PLA
        PLP
;
        JMP $EA31
```

Programa cargador en BASIC

```
 10 REM **** CARGADOR EN BASIC ****
 20 REM **** PARA UN SOLO SERVO ****
 30 :
 40 FOR I=820 TO 882
 50 READ A:POKE I,A
 60 CC=CC+A
 70 NEXT I
 80 READ CS:IF CC<>CS THEN PRINT
    "ERROR EN SUMA DE CONTROL":STOP
100 DATA120,173,20,3,174,196,3,141,196
110 DATA3,141,20,3,173,21,3,174,197,3
120 DATA141,197,3,142,21,3,88,96,8,72
130 DATA152,72,138,72,169,255,141,3
140 DATA221,160,255,136,208,253,172,0
150 DATA48,136,208,253,169,0,141,3,221
160 DATA104,170,104,168,104,40,76,49
170 DATA234
180 DATA7170:REM*SUMA DE CONTROL*
```

Programa de llamada en BASIC

```
 10 REM **** UN SOLO SERVOMOTOR ****
 20 :
 30 DN=8:REM SI CASSETTE ENTONCES
    DN=1
 40 IF A=0 THEN A=1:LOAD
    "SINGSERV.HEX",8,1
 50 POKE 964,79:POKE 965,3:REM APUNTAR
    A MANEJADOR IRQ
 60 RDD=56577:POKE RDD,255:REM TODAS
    SALIDA
 70 MC=820:SYS MC:REM ESTABLECER
    VECTOR IRQ
 80 POKE 53265,PEEK(53265)AND239:REM
    BORRAR PANTALLA
 90 :
100 GET K$:IF K$=" " THEN100:REM
    ESPERAR PULSACION TECLA
110 REM ** ALTERAR POSICION MOTOR **
120 IF ASC(K$)>48 AND ASC(K$)<58 THEN
    POKE 12288,VAL(K$)*20
130 IF K$<>"E" THEN 80:REM "E" PARA
    SALIR
140 END
```

Formas simétricas

El LOGO es un lenguaje particularmente útil para la investigación de patrones y simetrías

Existen cuatro clases de transformación que podemos aplicar a una figura bidimensional sin que su forma sufra ninguna modificación (aunque podría variar su posición). Estas transformaciones son: traslación, rotación, reflexión y reflexión con deslizamiento. Nuestro diagrama refleja cómo cambia la posición de una forma en función de cada una de estas transformaciones.

Se dice que una figura es *simétrica* si podemos transformarla en una o más de estas formas y dejar inalteradas tanto su posición como su forma. Las formas finitas (como los polígonos y las letras del alfabeto) deben tener simetrías basadas en la reflexión y la rotación, ya que las traslaciones y las reflexiones de deslizamiento cambiarán sus posiciones.

Para investigar estas simetrías resulta útil tener procedimientos en LOGO para hacer reflejar y rotar formas. Comenzaremos por analizar la tarea de reflejar una forma en una línea que pase por el origen y posea una orientación dada.

Es más sencillo si damos por sentado que el procedimiento para dibujar la forma es de estado transparente (es decir, deja a la tortuga en la misma posición y con la misma orientación que tenía antes de que se ejecutara el procedimiento). Nuestra tarea se divide entonces en dos partes: primero, hemos de hallar las coordenadas y la orientación del punto de partida de la reflexión que corresponda al punto de partida de la forma original. Es necesario llevar a cabo la segunda tarea antes de empezar a dibujar la forma. La misma supone cambiar todos los giros hacia la derecha del procedimiento para dibujar la forma por giros hacia la izquierda, y todos los giros hacia la izquierda por giros hacia la derecha. Una manera de hacerlo consiste en reemplazar todas las RT y LT del procedimiento mediante un procedimiento llamado GIRO, definido de la siguiente manera:

```
TO GIRO :A
  RT :DIR* :A
END
```

De modo que ahora podemos definir un cuadrado como:

```
REPEAT 4 [FD 50 GIRO 90]
```

Para utilizar este procedimiento debemos primero poner a 1 la variable local DIR. Por lo tanto, MAKE "DIR 1 CUADRADO dibujará un cuadrado. Para reflejar el cuadrado en el eje, todo lo que hay que hacer es digitar MAKE "DIR (−1) y luego CUADRADO. Pruébelo y vea lo que sucede.

El procedimiento para posicionar la tortuga antes de dibujar la reflexión depende en cierta medida de la trigonometría:

```
TO REFLEJAR :A
  MAKE "O ORIENTACION
  MAKE "XANT XCOR
  MAKE "ANGULO (ATAN :YANT :XANT)−90+:A
  MAKE "R SQRT (:XANT*:XANT+:YANT*:YANT)
  PU
  SETXY 0 0
  SETH :A + :ANGULO
  FD :R
  SETH 2* :A+:H
  PD
  MAKE "DIR :DIR*(−1)
END
```

Ahora este procedimiento se puede utilizar para ver el efecto de reflexiones en varias líneas a través del origen. Pruebe con:

```
MAKE "DIR 1
PU SETXY 40 70 PD
CUADRADO
REFLEJAR 60
CUADRADO
```

Si la forma reflejada se halla completamente en la parte superior del original, se dice que posee una "simetría reflexiva", respecto a esa línea. Pruebe con:

```
MAKE "DIR 1
PU SETXY 0 0 PD
CUADRADO
REFLEJAR 45
CUADRADO
```

Se podría escribir un procedimiento similar para hacer girar una forma respecto a un punto dado a través de un ángulo dado, pero eso lo dejaremos para que lo escriba usted mismo.

Algunos patrones, como los de los papeles pintados, utilizan en su diseño la misma forma repetidamente. Se pueden efectuar traslaciones y reflexiones con desplazamiento que muevan el patrón entero y, aun así, lo dejen exactamente tal como estaba. Por el momento nos concentraremos en patrones que impliquen traslaciones a lo largo de una única línea, dejando los patrones bidimensionales para el próximo capítulo.

Las combinaciones de las cuatro transformaciones fundamentales dan origen a apenas siete tipos de patrones sobre una línea recta. Todas estas posibilidades se muestran en nuestro segundo diagrama en forma de un simple motivo "PATA". Hemos construido procedimientos para dibujar los siete patrones a partir de cualquier MOTIVO empleando los procedimientos MOVER para traslación, GIRAR para rotación, e I.MOTIVO, que convierte todos los giros RT de MOTIVO en giros LT, y todos los giros LT en RT.

Hemos utilizado las facilidades para proceso de listas del LOGO para escribir el procedimiento I.MOTIVO reescribiendo MOTIVO. El procedimiento que empleamos para hacer esto es:

```
TO REESCRIBIR :PROC
  OUTPUT REESCRIBIR.PROC TEXT :PROC
END
```

REESCRIBIR toma el texto de un procedimiento especificado, lo modifica y lo crea bajo otro nombre. Da por sentado que el procedimiento sobre el cual está trabajando está escrito con primitivas del LOGO y no contiene ningún subprocedimiento. REESCRIBIR contiene una llamada a los siguientes procedimientos:

```
TO REESCRIBIR.PROC :TEXTO
  IF :TEXTO=[ ] THEN OUTPUT [ ]
  OUTPUT FPUT REESCRIBIR.LINEA FIRST :TEXTO
  REESCRIBIR.PROC BUTFIRST :TEXTO
END
```

Este procedimiento divide la tarea de reescribir el procedimiento de entrada en líneas individuales, mediante la llamada al siguiente procedimiento:

```
TO REESCRIBIR.LINEA :LINEA
  IF :LINEA=[ ] THEN OUTPUT [ ]
  IF LIST? FIRST :LINEA THEN OUTPUT FPUT
  REESCRIBIR.LINEA FIRST :LINEA REESCRIBIR.
  LINEA
  BUTFIRST :LINEA
  OUTPUT FPUT
  CAMBIAR.PALABRA FIRST :LINEA
  REESCRIBIR.LINEA BUTFIRST :LINEA
END
```

REESCRIBIR.LINEA realiza el proceso de cada línea, pasándole las palabras individuales a CAMBIAR.PALABRA para que ésta se encargue de ellas. La línea que comienza con IF LIST? es necesaria para tratar con una situación en la que MOTIVO contenga una sentencia REPEAT. Si usted no utiliza esta sentencia en sus procedimientos MOTIVO, entonces sí puede eliminar la línea de este procedimiento. El listado para CAMBIAR.PALABRA es:

```
TO CAMBIAR.PALABRA :PALABRA
  IF (ANYOF :PALABRA="RT :PALABRA=
  "RIGHT) THEN OUTPUT "LEFT
  IF (ANYOF :PALABRA="LT :PALABRA=
  "LEFT) THEN OUTPUT "RIGHT
  OUTPUT :PALABRA
END
```

Este procedimiento verifica cada palabra individual y realiza las modificaciones necesarias. Habiendo entrado todos estos procedimientos, veamos ahora cómo funcionan. En primer lugar, es preciso definir una forma simple, como por ejemplo:

```
TO TRI
  REPEAT 3 [FD 50 RT 120]
END
```

Ahora entre DEFINE "REF REESCRIBIR "TRI, y llame a REF. Su definición ha de ser:

```
TO REF
  REPEAT 3 [FD 50 LEFT 120]
END
```

Es bastante factible escribir un procedimiento REESCRIBIR más general que también reescriba cualquier procedimiento llamado por el procedimiento principal. Si intenta escribirlo, ¡tenga cuidado con los procedimientos recursivos! También necesitará poder comprobar si una palabra es el nombre de un procedimiento.

Los siete patrones de franjas

Sería posible (y elegante desde el punto de vista matemático) construir los patrones a partir de procedimientos para las cuatro transformaciones básicas. Los procedimientos para dibujar patrones hacen uso de estos tres procedimientos auxiliares:

```
TO POSICION
  HT
  PU
  SETXY -125 0
  PD
END
```

Éste posiciona la tortuga en el lado izquierdo de la pantalla, lista para empezar a dibujar.

```
TO MOVER
  PU
  RT 90
  FD 50
  LT 90
  PD
END
```

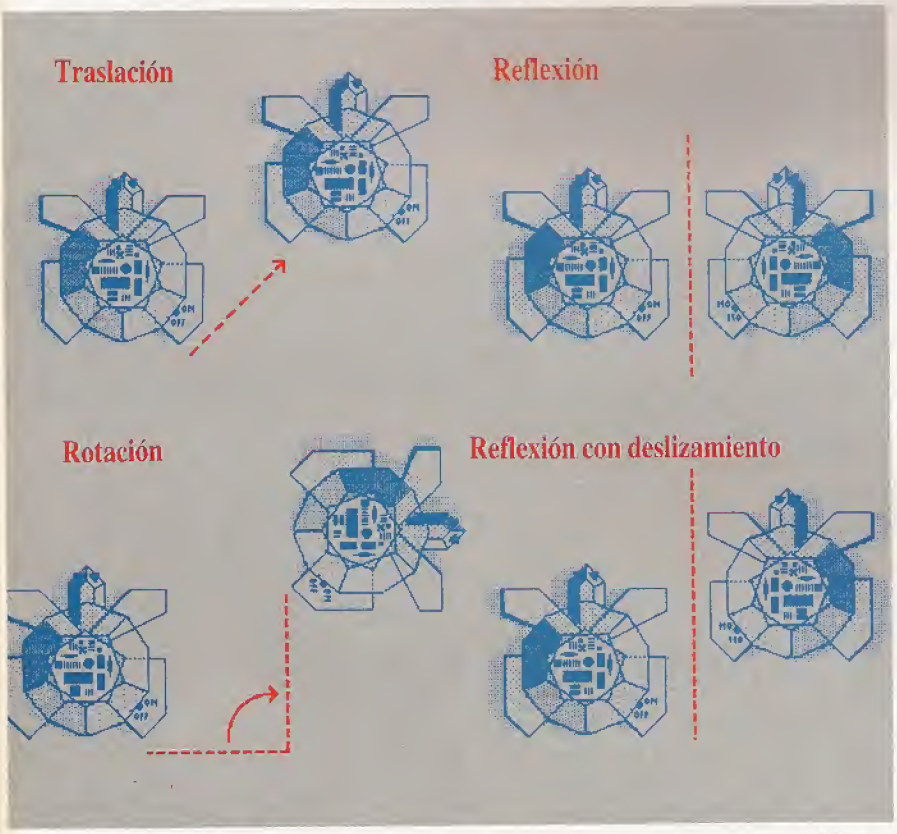
MOVER realiza la traslación requerida.

```
TO GIRAR
  RT 180
END
```

GIRAR efectúa la rotación que necesitamos.

Para utilizar estos procedimientos defina primero un procedimiento de forma (digamos, FORMA) que sea de estado transparente y que no incluya llamadas a subprocedimientos. Después ya puede dibujar el primer patrón empleando FORMA como su motivo, entrando PATRON1 "FORMA.

Transformaciones isométricas
Las transformaciones que alteran la posición de un objeto pero no su forma se llaman isometrías. Existen cuatro tipos básicos de transformaciones isométricas: traslación, rotación, reflexión y reflexión con deslizamiento. La traslación es un simple "deslizamiento" de la figura original. La rotación hace girar la forma alrededor de algún punto central especificado. La reflexión implica el movimiento de puntos a través de una línea de espejo, de modo que cada punto de la forma final esté a la misma distancia hacia un lado de la línea que el punto correspondiente del original hacia el otro lado. La reflexión con deslizamiento es una reflexión y una traslación. Mientras que la traslación y la rotación conservan el "sentido", la reflexión y la reflexión con deslizamiento lo cambian: imagínese, por ejemplo, una palabra reflejada en un espejo



Ian McKinnell

Logomotivo

Siete patrones de franjas

Para ejecutar los procedimientos de patrones debe tener en el espacio de trabajo los siguientes procedimientos: REESCRIBIR.PROC, REESCRIBIR.LINEA, CAMBIAR.PALABRA, POSICION, MOVER y GIRAR. Los siete patrones posibles son:

```
TO PATRON1 :PROC
  DEFINE "MOTIVO TEXT :PROC
  POSICION
  REPEAT 6 [MOTIVO MOVER]
END

TO PATRON2 :PROC
  DEFINE "MOTIVO TEXT :PROC
  DEFINE "I.MOTIVO REESCRIBIR.PROC
  TEXT :PROC
  POSICION
  REPEAT 3 [MOTIVO MOVER GIRAR
  I.MOTIVO GIRAR MOVER]
END

TO PATRON3 :PROC
  DEFINE "MOTIVO TEXT :PROC
  DEFINE "I.MOTIVO REESCRIBIR.PROC
  TEXT :PROC
  POSICION
  REPEAT 6 [MOTIVO I.MOTIVO MOVER]
END

TO PATRON4 :PROC
  DEFINE "MOTIVO TEXT :PROC
  DEFINE "I.MOTIVO REESCRIBIR.PROC
  TEXT :PROC
  POSICION
  REPEAT 6 [MOTIVO GIRAR MOTIVO
  GIRAR MOVER]
END

TO PATRON5 :PROC
  DEFINE "MOTIVO TEXT :PROC
  DEFINE "I.MOTIVO REESCRIBIR.PROC
  TEXT :PROC
  POSICION
  REPEAT 3 [MOTIVO I.MOTIVO MOVER
  GIRAR MOTIVO I.MOTIVO GIRAR
  MOVER]
END

TO PATRON6 :PROC
  DEFINE "MOTIVO TEXT :PROC
  DEFINE "I.MOTIVO REESCRIBIR.PROC
  TEXT :PROC
  POSICION
  REPEAT 6 [MOTIVO GIRAR I.MOTIVO
  GIRAR MOVER]
END

TO PATRON7 :PROC
  DEFINE "MOTIVO TEXT :PROC
  DEFINE "I.MOTIVO REESCRIBIR.PROC
  TEXT :PROC
  POSICION
  REPEAT 6 [MOTIVO I.MOTIVO GIRAR
  MOTIVO I.MOTIVO GIRAR MOVER]
END
```

Es necesario ejecutar estos procedimientos con algún procedimiento para dibujar un motivo adecuado. El motivo que hemos utilizado es:

```
TO PATA
  FD 50
  RT 90
  FD 20
  BK 20
  LT 90
  BK 50
END
```

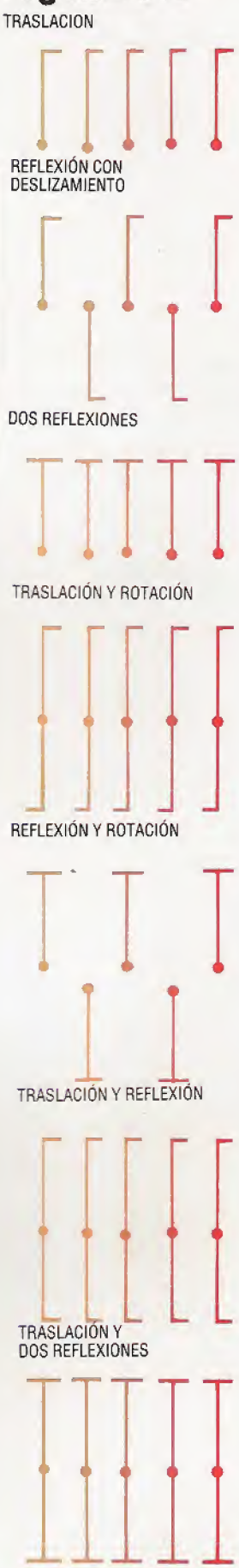
Un motivo alternativo es:

```
TO FIG
  RT 30
  FD 20
  LT 50
  FD 20
  RT 90
  FD 10
  REPEAT 4 [FD 20 RT 90]
  BK 10
  LT 90
  BK 20
  RT 50
  BK 20
  LT 30
END
```

Complementos al LOGO

ATAN y TOWARDS no existen en el LOGO Atari, ni tampoco existe un recambio sencillo. Ello incide en los procedimientos REFLEXION y ROTACION, pero no en los PATRON.

El LOGO Atari no posee TEXT ni DEFINE como primitivas, si bien el manual ofrece un método para definirlas. Usted puede escribir I.MOTIVO modificando MOTIVO mediante el editor



Siete de la misma clase
Las cuatro transformaciones isométricas primitivas se pueden combinar de diversas formas para producir siete patrones exclusivos, tal como vemos en la ilustración. En cada caso partimos del motivo "pata" y todas las traslaciones se efectúan en la dirección del eje *x*

Liz Dixon

Jaque al chip

En el Commodore 64 es fácil reubicar posiciones de memoria. Lo demostraremos con una rutina para diseñar y almacenar un máximo de ocho visualizaciones

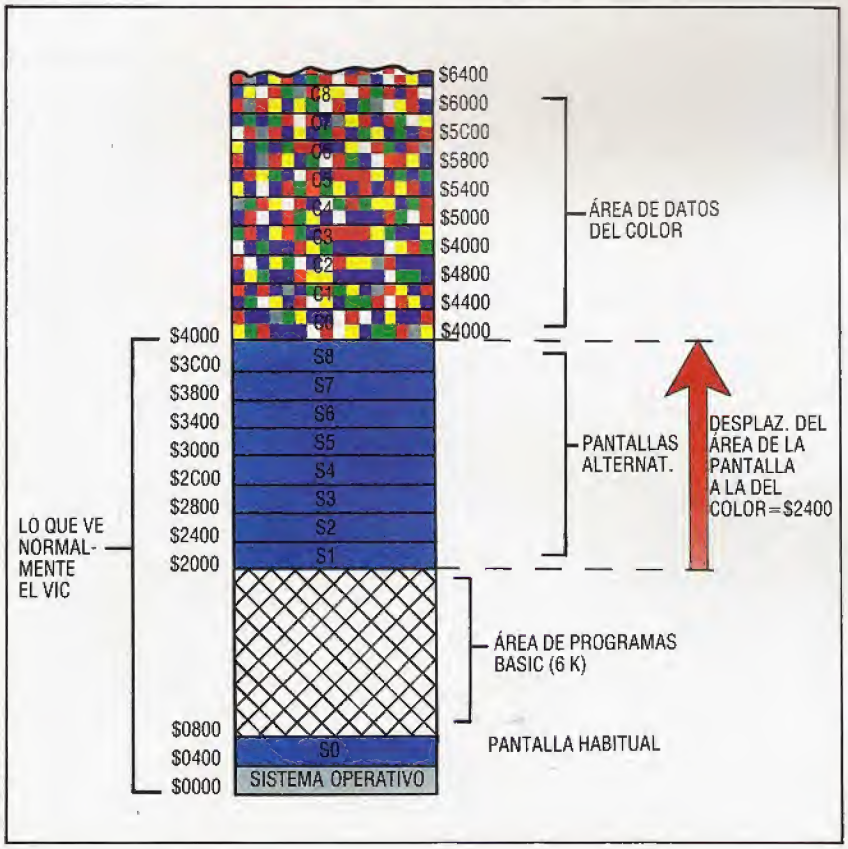
Panorama VIC
El chip controlador de video (VIC) del Commodore 64 puede "ver" 16 K de memoria. Generalmente se trata de los primeros 16 K, del $0000 al $3FFF, pero puede hacerse que "mire" hacia cualquier otro de los tres bloques de 16 K, alterando el contenido de uno de los registros de control del VIC. El programa "Pantallas alternativas" establece hasta nueve mapas de pantalla dentro del área normal de 16 K que puede ver el chip. Los mapas correspondientes a cada pantalla se encuentran en el área de la RAM que está justo encima del área vista por el VIC, disponiendo cada pantalla, salvo la pantalla 0, de un desplazamiento constante de $2400 respecto a su mapa de color

En el Commodore 64, un chip especial controla la visualización y el tratamiento de sprites. Es el chip denominado VIC II. El chip VIC accede a diversos sectores de la memoria para obtener información útil para crear la visualización que se nos presenta en la pantalla. Estas áreas incluyen la ROM de caracteres, que contiene la forma del trazado de los caracteres; la RAM del color, que guarda la información del color en pantalla, y la RAM de la pantalla. Esta última contiene información de los caracteres que pueden visualizarse en las 1 000 posiciones (25 filas por 40 columnas) que constituyen el espacio de la pantalla.

En el momento de conectar el Commodore 64, el chip VIC supone que la pantalla está posicionada en esos 1 000 bytes que comienzan en la posición 1024 ($0400) y accede a esta área para buscar la información inicial de la pantalla. Pero si se altera el valor de un registro dentro del chip VIC es posible redireccionar el citado chip a otra zona de memoria, normalmente en los primeros 16 K de la memoria. Los cuatro bits superiores del registro de control en el VIC se encuentran en la posición 53272 ($D018) y determinan el bloque de K que ha de ser tomado como la pantalla, dentro de la citada área de 16 K. He aquí un cuadro que ilustra los valores de los bits que corresponden a cada una de las 16 posibles posiciones de pantalla:

Patrón de bits	Inicio pantalla	
0000XXXX	0	$0000
0001XXXX	1024	$0400 *
0010XXXX	2048	$0800
0011XXXX	3072	$0C00
0100XXXX	4096	$1000
0101XXXX	5120	$1400
0110XXXX	6144	$1800
0111XXXX	7168	$1C00
1000XXXX	8192	$2000
1001XXXX	9216	$2400
1010XXXX	10240	$2800
1011XXXX	11264	$2C00
1100XXXX	12288	$3000
1101XXXX	13312	$3400
1110XXXX	14336	$3800
1111XXXX	15360	$3C00

*=Posición por defecto



Kevin Jones

Para que el chip VIC traslade la pantalla a otra área, hemos de cambiar los cuatro bits superiores de la posición 53272 ($D018) por los valores indicados en este cuadro. Los cuatro bits inferiores no deben ser modificados (los hemos indicado con XXXX), pues controlan otra función. Para poner a cero los cuatro bits superiores sin cambiar el valor de los inferiores, debemos operar el contenido del registro con un AND lógico empleando el número 15 (00001111 en binario). Una vez hecho esto operaremos con OR el nuevo contenido del registro empleando el valor que deseamos. Para colocar la pantalla en la última área que el chip puede ver (es decir, la que comienza en 15360, o sea, $3C00), habremos de emplear un OR con el contenido del registro y el número 240 (binario, 11110000). En BASIC se encarga de todo esto la siguiente instrucción POKE:

```
POKE 53272,(PEEK(53272)AND15)OR240
```

Antes de poder escribir algo en nuestra nueva pantalla, tendremos también que decirle al sistema operativo del Commodore 64 que la posición de la pantalla fue alterada. Lo que conseguimos colocando el byte *hi* de la nueva dirección de inicio de la pantalla en la posición 648 ($0288). Para la pantalla más alta resulta ser $3C, y se obtiene fácilmente en BASIC dividiendo por 256 la dirección de inicio de pantalla.

Una vez cambiado el contenido de estos dos re-

gistros podemos hacer uso de la nueva pantalla como si nada hubiera ocurrido. Observe que si hubiera que cambiar la posición de la pantalla en BASIC habría que escribir un pequeño programa para hacerlo.

Es posible hacer uso de la facilidad con que en un Commodore 64 puede moverse la pantalla para obtener varias utilidades de interés. En concreto, podemos cambiar la visualización con rapidez y sin dificultad. Sólo que si deseamos mover la pantalla hay que mover también la RAM correspondiente del color, ya que la visualización quedará sin gracia, faltándole los datos adecuados de la RAM del color. A pesar de que podemos establecer varias áreas dentro de la memoria y cambiarlas para pantalla, sólo se dispone de una inamovible área RAM del color, lo que significa que, para retener varias visualizaciones individuales de pantalla, debemos reservar zonas de memoria capaces de contener los 1 000 bytes de datos de color para cada pantalla. Cuando se desea visualizar una pantalla hay que copiar la información en la RAM del color y guardar los datos en una de las áreas de la memoria escogidas para albergar la RAM del color, antes de cambiar a otra pantalla diferente.

Organización de la memoria

La tarea principal de esta utilidad es organizar la memoria para pantallas alternas (junto con sus correspondientes áreas para datos del color), y realizar la transferencia de bloques de memoria. Dado que el chip VIC puede "ver" 16 K de memoria, podemos diseñar un sistema que nos permita disponer hasta de ocho pantallas diferentes y un programa BASIC a medida. En un diagrama adjunto hemos ilustrado la organización de la memoria que emplea la utilidad.

Para estar seguros de que no va a ser machacada ningún área de pantalla o de color a causa de algún programa BASIC, debemos bajar la frontera superior de la memoria BASIC. Esto lo hace la siguiente instrucción en nuestro programa en BASIC:

```
POKE 55,0:POKE 56,32:CLR
```

La dirección de base de cualquier pantalla se calcula a partir de su número mediante esta fórmula:

Base pantalla = $1C00 + ($0400*Número pantalla)

La dirección de base de la correspondiente área del color se calculará añadiendo un desplazamiento a la dirección de base de la pantalla. La fórmula es:

Base color = $2400 + Base pantalla

Las áreas del color pueden ubicarse en cualquier lugar de la RAM, pero es recomendable colocarlas justo encima de la última pantalla visible para el chip VIC. Observe que está incluida un área de color para la pantalla 0, que es la pantalla habitual. Para esta particular área de color, el desplazamiento ha de ser diferente y deberá ser tenido en cuenta en nuestro programa. El registro de control del VIC y el registro del sistema operativo pueden establecerse para cualquier pantalla mediante:

Registro VIC = $70 + ($10*Número pantalla)
Registro OS = Byte HI de la dirección base de pantalla

Además, para manejar el establecimiento de registros y realizar las transferencias adecuadas de los datos de color desde y hacia el área de RAM de color, el programa almacena también el color del fondo y el borde o recuadro de la pantalla. Estas dos tareas se controlan mediante un par de registros en el chip VIC: 53280 ($D020) controla el color del borde o recuadro (*border*) y 53281 ($D021) controla el color del fondo de la pantalla (*paper*). La utilidad establece una tabla dentro de su área para almacenar estos atributos para cada pantalla.

Modos operativos

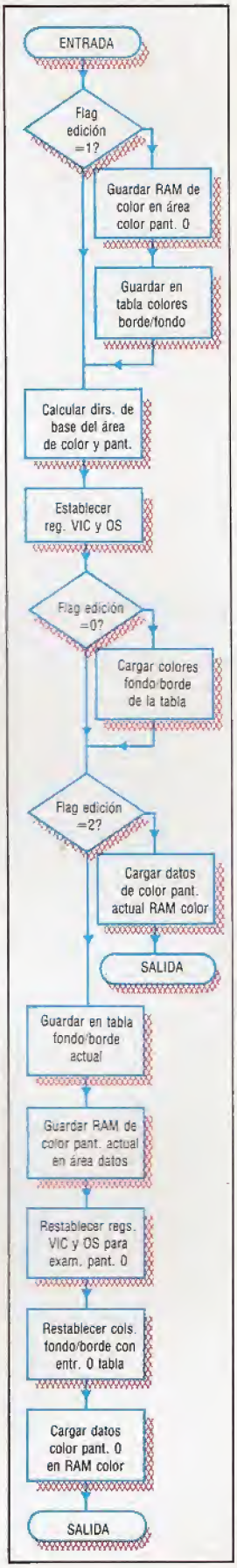
La utilidad tiene dos modos de operar: puede editar(*edit*) o visualizar (*display*) una pantalla seleccionada. En cada caso el número de pantalla a emplear debe colocarse (POKE) en 49152 ($C000). Para conseguir que sea la misma llamada SYS nos servimos de un flag especial que indicará cuál ha sido el modo escogido. Este flag se coloca (POKE) en la posición 49153 ($C001).

0: indica el modo visualización
1: indica el modo edición

El modo edición funciona de forma poco habitual, para poder emplear todas las facilidades del editor de pantalla (como cambio de color de texto, establecimiento de modo invertido y borrado de pantalla). La utilidad debe ser llamada antes y poner a 1 el flag de edición. Después guardará el área de color normal de pantalla y los colores de papel y recuadro, pondrá el valor adecuado en los registros del sistema operativo y del VIC. En este punto, el programa en BASIC de llamada toma el control poniendo el cursor en su inicio y empleando la instrucción INPUT del BASIC. Esta instrucción esperará un carácter de retorno (13 en ASCII) antes de proseguir. Mientras, todas las funciones del editor de pantalla pueden ser usadas de la manera habitual para editar la pantalla seleccionada; cuando la tarea esté concluida se pulsará Return.

El programa BASIC debe posteriormente llamar por segunda vez a la utilidad, pero en esta ocasión el flag de edición se pone a 2. Esto hace que se guarden los datos del color y los colores del fondo-/borde de la pantalla con los que se ha trabajado antes de restablecer la pantalla habitual. Si se desea cambiar el color del borde o del fondo, estos colores deben colocarse (POKE) en las posiciones 49154 ($C002) y 49155 ($C003) respectivamente. Aunque la rutina está diseñada para esta tarea concreta, incorpora además rutinas generales para establecer registros de control y copiar los datos desde o hacia la RAM del color. No le será, por tanto, difícil idear una utilidad para sus propias especificaciones partiendo de las rutinas generales.

El programa de llamada en BASIC está diseñado para visualizar un menú que dé la posibilidad de editar o visualizar. La rutina de visualización llama a cada una de las ocho pantallas diferentes secuencialmente y respondiendo a pulsaciones del teclado. Las pantallas seguirán rotando hasta que se pulse la tecla Return. Entonces se restaurará la pantalla normal y se volverá al menú. Si se escoge la opción de edición, el usuario puede establecer los colores del fondo y del recuadro para la pantalla escogida, y puede servirse del editor de pantalla de la manera habitual para modificar el contenido de ésta. Una vez concluido, Return hace que el cuadro se almacene y se restablezca la pantalla normal.



Liz Dixon

Cargador de BASIC

```
10 REM ************************************
20 REM **        CARGADOR DE BASIC PARA  **
30 REM **        PANTALLAS ALTERNATIVAS  **
40 REM ************************************
50 :
60 FOR I=49152 TO 49439
65 READA:POKE I,A
70 CC=CC+A
75 NEXT
80 READ CS:IFCS<>CC THENPRINT"CHECKSUM ERROR":STOP
100 DATA255,255,0,0,191,255,0,0,255
110 DATA191,0,0,191,191,0,0,255,191,0
120 DATA0,255,191,0,0,255,255,0,173,1
130 DATA192,201,1,208,30,169,0,141,6
140 DATA192,169,64,141,7,192,32,224
150 DATA192,162,0,32,246,192,173,2,192
160 DATA141,32,208,173,3,192,141,33
170 DATA208,173,0,192,208,6,32,32,193
180 DATA76,138,192,169,0,141,4,192,141
190 DATA5,192,174,0,192,173,5,192,24
200 DATA105,4,202,208,250,24,105,28
210 DATA141,5,192,24,105,36,141,7,192
220 DATA173,0,192,162,4,10,202,208,252
230 DATA24,105,112,141,8,192,173,24
240 DATA208,41,15,13,8,192,141,24,208
250 DATA173,5,192,141,136,2,173,1,192
260 DATA208,5,174,0,192,32,211,192,201
270 DATA2,240,4,32,189,192,96,174,0
280 DATA192,32,246,192,32,224,192,32
290 DATA202,193,169,0,141,6,192,169,64
300 DATA141,7,192,162,0,32,211,192,32
310 DATA189,192,96,173,6,192,133,251
320 DATA173,7,192,133,252,169,0,133
330 DATA253,169,216,133,254,32,3,193
340 DATA96,169,9,192,141,32,208,189,18
350 DATA192,141,33,208,96,173,6,192
360 DATA133,253,173,7,192,133,254,169
370 DATA0,133,251,169,216,133,252,32,3
380 DATA193,96,173,32,208,157,9,192
390 DATA173,33,208,157,18,192,96,162,3
400 DATA160,0,177,251,145,253,136,208
410 DATA249,230,252,230,254,202,48,10
420 DATA208,240,177,251,145,253,160
430 DATA231,208,232,96
440 DATA36606:REM "CHECKSUM"
```

Pantallas alternativas

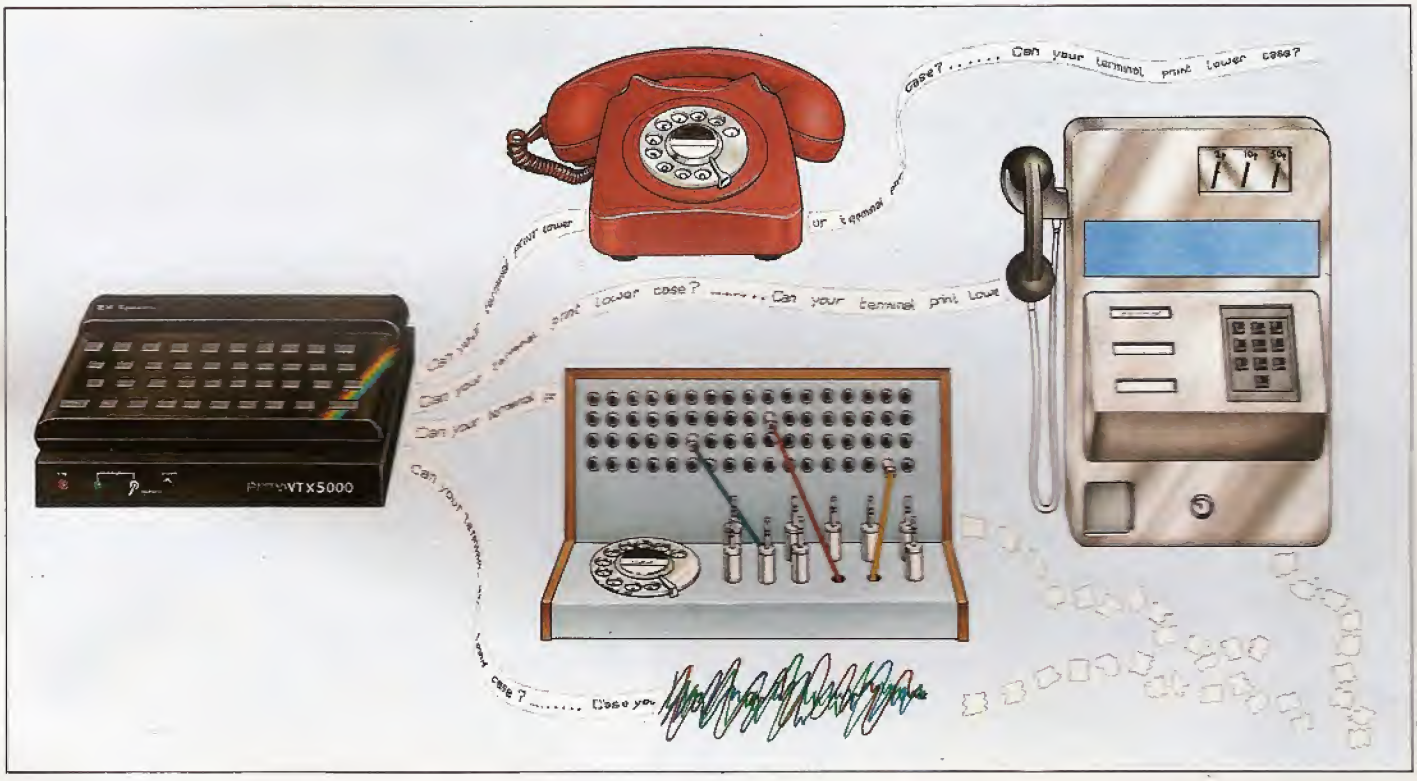
```
  3 REM ************************************************
  5 REM **                                            **
 10 REM **              PANTALLAS ALTERNATIVAS        **
 11 REM **                                            **
 12 REM ************************************************
 13 :
 15 DN=8:REM PARA CASSETTE DN=1
 20 IF A=0 THEN A=1:LOAD "ALT SCREENS.HEX",DN,1
 25 POKE55,0:POKE56,32:CLR:REM BAJA FRONTERA MEMORIA
 40 SCNUMB=49152: REM NUMERO PANTALLA (SCREEN)
 70 EDITFG=49153: REM 0=VISUALIZA PANTALLA
 75 :              REM 1=EDITA PANTALLA
 77 :              REM 2=COPIA RAM COLOR
 80 ALT=49179:     REM DIRECCION INICIO EN COD MAQ
 85 BRDCOL=49154:REM COLOR BORDE
 87 PAPCOL=49155:REM COLOR FONDO (PAPER)
 90 :
 95 REM **** MENU PRINCIPAL ****
 96 :
100 PRINTCHR$(147):REM LIMPIA PANTALLA
105 PRINTCHR$(154):REM LETRAS AZULES LT
110 DN$="      ":REM Q=CARACT CURSOR ABAJO
130 PRINTTAB(8);DN$;"CBM64 PANTALLAS ALTERN"
135 PRINTTAB(8);"
140 PRINTTAB(5);DN$;"F1 — EDITA DIBUJO"
150 PRINTTAB(5);DN$;"F3 — VISUALIZA SECUENCIA DIBUJO"
200 :
210 GETAS:IFAS=" " THEN 210:REM ESPERA PULSACION TECLA
220 IFAS="  " THEN GOSUB 1000
230 IFAS="  " THEN GOSUB 1500
260 GOTO 100
999 :
1000 REM **** EDITA PANTALLA ****
1002 :
1005 EF=1:REM ACTIVA MODO EDICION
1010 PRINTCHR$(147)
1020 PRINTTAB(16);DN$;"MODO EDICION"
1030 PRINTDN$;:INPUT"NUM.PANTALLA";SNS
1050 IFASC(SNS)<48 OR ASC(SNS)>56 THEN 1030
1060 PRINT DN$; :INPUT "COLOR BORDES";BCS
1070 IFVAL(BCS)=0 AND BCS<>"0" THEN 1060
1080 PRINT DN$; :INPUT "COLOR FONDO";PCS
1090 IF VAL(PCS)=0 AND PCS<>"0" THEN 1080
1100 :
1110 POKEEDITFG,EF
1120 POKESCNUMB,VAL(SNS)
1130 POKEBRDCOL,VAL(BCS)
1140 POKEPAPCOL,VAL(PCS)
1150 SYS ALT
1155 REM ** ESPERA RETURN **
1162 INPUT "    ";XS:REM S=CURSOR EN INICIO
1165 REM ** GUARDA PANTALLA **
1170 EF=2
1175 POKE EDITFG,EF
1180 SYS ALT
1185 RETURN
1190 :
1500 REM **** VISUALIZA PANTALLA ****
1510 EF=0
1520 PRINT CHR$(147)
1545 SN=1
1550 POKE EDITFG,EF
1555 POKE SCNUMB,SN
1560 SYS ALT
1570 GET X$: IFX$=" " THEN1570:REM ESPERA PULSACION TECLA
1572 IF X$=CHR$(13) THEN 1580:REM PANTALLA HABITUAL
1575 SN=SN+1:IF SN<9 THEN 1555
1577 SN=1: GOTO 1555
1580 POKESCNUMB,0:SYS ALT
1600 RETURN
```

Commodore 64

```
;++++++++++++++++++++
;++++++++++++++++++++
;++        PANTALLAS       ++
;++      ALTERNATIVAS      ++
;++      PARA EL CBM 64    ++
;++++++++++++++++++++
;++++++++++++++++++++
;
FROM    =$FB            ;PAGINA 0
TO      =$FD            ;PUNTEROS
;
CRAMLO=$00              ;INICIO RAM COLOR (CRAM)
CRAMHI=$D8
NCOLLO=$00              ;COLOR HABITUAL
NCOLHI= $40             ;DIRECCION BASE
NSCRHI= $04             ;DIR BASE PANTALLA HABITUAL
NVCPOK=$10              ;VALOR HABITUAL REG VIC
;
BLOCKS=$03              ;NUM DE BLOQUES DE 256 BYTES
EXTRA   =$E7            ;CANT EXTRA DE 1000 BYTES
;
COLOFF= $24             ;BYTE HI DESPL COLOR
SCROFF= $1C             ;BYTE HI DESPL PANTALLA
VICOFF= $70             ;DESPL REG CONTROL DEL VIC
VCMASK=$0F              ;MASCARA BYTE LO REG CONTROL VIC
VICREG= $D018           ;REG CONTROL POSICION PANT
EDREG=$0288             ;REG NUCLEO EDICION PANT
BORDER=$D020            ;REG COLOR BORDE
PAPER   =$D021          ;REG COLOR FONDO
;
*=$C000                 ;EST PUNTERO CARGA
;
SCNUMB  *=*+1           ;NUMERO PANTALLA
EDITFG  *=*+1           ;FLAG MODO EDICION
BRDCOL  *=*+1           ;COLOR BORDE
PAPCOL  *=*+1           ;COLOR FONDO
;
SCBASE  *=*+2           ;ALMAC BASE PANTALLA
CLBASE  *=*+2           ;ALMAC BASE COLOR
VPOKE   *=*+1           ;ALMAC PROVIS PARA NUMERO VIC
BRDTAB  *=*+9           ;TABLA COLORES BORDE
PAPTAB  *=*+9           ;TABLA COLORES FONDO
;
;**** GUARDA PANTALLA 0 ****
        LDA EDITFG
        CMP #$01        ;SI ES 0 O BIEN 2
        BNE CALC        ;ENTONCES NO SE GUARDA
;
        LDA #NCOLLO
        STA CLBASE
        LDA #NCOLHI
        STA CLBASE+1
        JSR SAVE        ;GUARDA CRAM EN C0
;
        LDX #$00
        JSR SAVEBP      ;GUARDA REGISTROS B/P
;
        LDA BRDCOL
        STA BORDER      ;ESTABLECE NUEVOS B/P
        LDA PAPCOL      ;COLORES
        STA PAPER
;
;**** CALCULA BASE COLOR ****
;
CALC
        LDA SCNUMB
        BNE NOZERO
        JSR RESET       ;EST REGISTROS NORMALES
        JMP TESTIFG
;
NOZERO
        LDA #$00
        STA SCBASE
        STA SCBASE+1    ;INICIO BASE PANT
;
        LDX SCNUMB      ;CARGA NUM PANTALLA
        LDA SCBASE+1    ;SOLO BYTE HI
MULT
        CLC
        ADC #$04
        DEX
        BNE MULT
;
        CLC
        ADC #SCROFF     ;SUMA DESPL PANTALLA
        STA SCBASE+
;
        CLC
        ADC #COLOFF     ;SUMA DESPL COLOR
        STA CLBASE+1
;
;**** EST REGS VIC Y EDITOR ****
        LDA SCNUMB
        LDX #$04
MORE
        ASL A
        DEX
        BNE MORE        ;MULT POR 16
;
        CLC
        ADC #VICOFF     ;SUMA DESPL
        STA VPOKE
        LDA VICREG
        AND #VCMASK
        ORA VPOKE
        STA VICREG      ;EST REG VIC
;
        LDA SCBASE+1
        STA EDREG       ;EST REG EDITOR
;
;**** COMPROBACION ESTADO FLAG EDICION ****
TESTFG
        LDA EDITFG
        BNE NOLOAD      ;SI 0 MODO VISUALIZ
        LDX SCNUMB
        JSR LOADBP      ;CARGA B/P EN REGS
;
;
NOLOAD
        CMP #$02
        BEQ CONT        ;SI 2 ENTONCES GUARDA RAM
;
        JRS LOAD        ;CARGA COLOR EN GRAM
        RTS
;
CONT
        LDX SCNUMB
        JSR SAVEBP      ;GUARDA REGS BP
        JSR SAVE        ;GUARDA CRAM EN COL
        JSR RESET       ;EST REGS NORMALES
        LDA #NCOLLO
        STA CLBASE      ;CARGA CBASE CON BASE C0
        LDA #NCOLHI
        STA CLBASE+1
        LDX #$00
        JSR LOADBP      ;RECARGA COLORES INIC BORDE/FONDO
        JSR LOAD        ;CARGA COLORES PANT NORMAL
        RTS
;
;**** S/R DE TRANSFERENCIA A LA RAM ****
LOAD
        LDA CLBASE
        STA FROM        ;CARGA PUNTEROS PAGINA 0
        LDA CLBASE+1
        STA FROM+1
;
        LDA #CRAMLO
        STA TO
        LDA #CRAMHI
        STA TO+1
;
        JSR COPY        ;COPIA AREA RAM
        RTS
;**** S/R DE CARGA COLS. BORDE/FONDO ****
LOADBP
        LDA BRDTAB,X
        STA BORDER
        LDA PAPTAB,X
        STA PAPER
        RTS
;
;**** S/R DE TRANSFERENCIA DESDE RAM ****
SAVE
        LDA CLBASE
        STA TO          ;CARGA PUNTEROS PAG 0
        LDA CLBASE+1
        STA TO+1
;
        LDA #CRAMLO
        STA FROM
        LDA #CRAMHI
        STA FROM+1
;
        JSR COPY        ;COPIA AREA RAM
        RTS
;
;**** S/R DE ALMAC COLORES BORDE/FONDO ****
SAVEBP
        LDA BORDER
        STA BRDTAB,X
        LDA PAPER
        STA PAPTAB,X
        RTS
;
;**** S/R DE COPIA 1000 BYTES ****
;
COPY
        LDX #BLOCKS
        LDY #$00
NEXT
        LDA (FROM),Y
        STA (TO),Y
        DEY
        BNE NEXT
NXBLOC
        INC FROM+1
        INC TO+1
        DEX
        BMI FINISH
        BNE NEXT
        LDA (FROM),Y
        STA (TO),Y
        LDY #EXTRA      ;BYTES EXTRA
        BNE NEXT
FINISH
        RTS
;
;**** S/R REST REGS VIC Y EDICION ****
RESET
        LDA #NCOLLO
        STA CLBASE
        LDA #NCOLHI
        STA CLBASE+1
        LDA VICREG
        AND #VCMASK
        ORA #NVCPOK
        STA VICREG      ;RESTAURA REG VIC
        LDA #NSCRHI
        STA EDREG       ;RESTAURA REG ED
        RTS
```

Mensaje recibido

En este capítulo analizaremos los protocolos que se precisan para que dos ordenadores se comuniquen a través de un modem



Steve Cross

El término "dúplex" nos indica si un sistema puede o no transmitir y recibir datos al mismo tiempo. Un sistema dúplex total *(full-duplex)* permite la transferencia bidireccional de datos, mientras que los sistemas medio dúplex *(half-duplex)* no lo permiten. Esto se puede comparar con la diferencia que existe entre hablar por teléfono, con dos interlocutores que pueden hablar a la vez, y utilizar un radio-teléfono, en el cual la pulsación del botón de transmisión impide automáticamente la recepción.

La principal ventaja de un sistema dúplex-total es su capacidad para interrumpir al ordenador transmisor. Si, por ejemplo, se está transmitiendo un menú largo y usted ya sabe que desea la opción 1, puede pulsar "1" y el sistema actuará de inmediato sobre esa entrada. En un sistema medio dúplex, tendría que esperar hasta que se transmitiera el menú completo antes de efectuar su selección.

El dúplex total se obtiene mediante el empleo de frecuencias distintas para cada máquina. El ordenador que hace la llamada utilizada una frecuencia que se conoce como la «de origen», mientras que el ordenador anfitrión utiliza la frecuencia «de respuesta».

Protocolos de terminal

Cuando un ordenador transmite datos a otro a través de una línea telefónica, inicialmente no sabe nada acerca del ordenador del extremo receptor. El terminal puede ser un Spectrum con una visualización de 32 columnas o un teletipo de 132 columnas. Existe la posibilidad de que tenga o no color. De que soporte o no caracteres en minúscula. En resumen, puede tratarse de cualquier máquina, desde un micro hasta un ordenador central.

Existen tres formas de abordar este problema. La primera es el enfoque seguido por el lenguaje BASICODE, que adopta el mínimo denominador común y transmite sólo aquellos formatos que puedan ser manipulados por el terminal equipado más elementalmente. La segunda consiste en solicitar información respecto a las capacidades del terminal dado y luego modificar la salida para adecuarla a ellas. El otro enfoque es aquel en el cual el sistema da por sentado que ha accedido a un determinado terminal, o a un tipo de terminal, y deja que el software del usuario haga frente a los datos transmitidos.

El primer enfoque se aplica muy raramente, dado que el mínimo denominador común entre terminales es el teletipo estándar. Éste sólo posee un juego de caracteres en mayúscula, carece de formateado de impresión, de color y de gráficos y es de poca velocidad. El segundo procedimiento es más común: es el que utilizan la mayoría de los tablones de anuncios. El tercer método es el que suelen emplear los sistemas comerciales y universitarios, y se

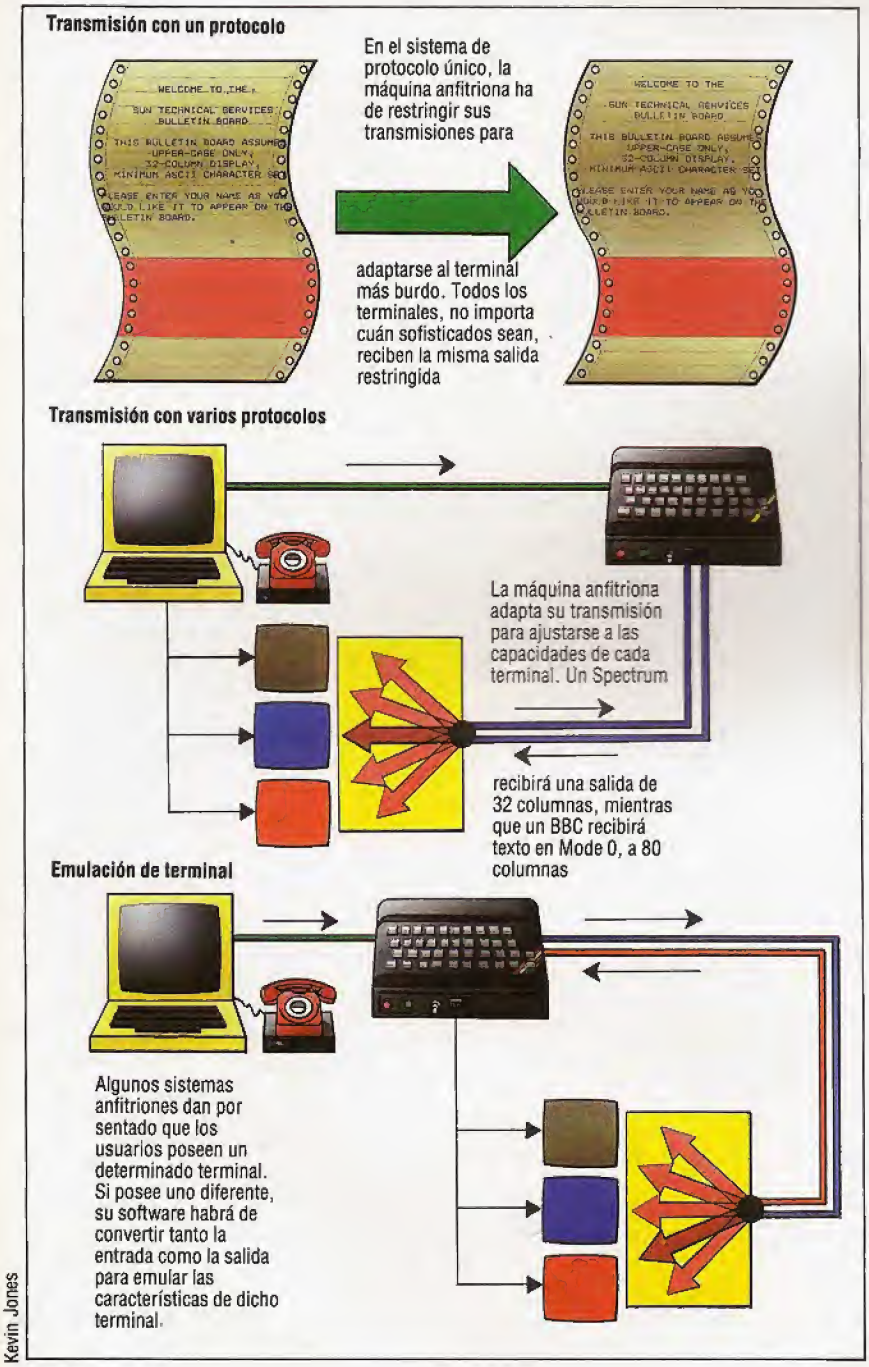
Fuera de lugar
En los anuncios publicitarios se suelen mostrar micros portátiles y moderns acústicos que operan a pilas funcionando en toda clase de lugares; pero existen algunas situaciones en que *no* pueden ser utilizados. Algunos ejemplos en este sentido son los teléfonos públicos comunes, las centralitas telefónicas manuales (que se emplean en muchos hoteles) y en las líneas de baja calidad y con muchas interferencias, problema éste que afecta a numerosas zonas apartadas

Observando el protocolo
Existen tres enfoques principales para los protocolos de comunicaciones. El primero adopta el mínimo denominador común (velocidad de transmisión reducida, pocas columnas de texto, ausencia de color y de gráficos, etc.). En el segundo la máquina anfitriona modifica su salida con el fin de adaptarse a diversos terminales (técnica que aplican la mayoría de los tablones de anuncios). El tercero es para que el terminal se "comporte" como un determinado terminal; esto se consigue mediante una técnica denominada *emulación de terminal*

basa en una técnica de software que se conoce como *emulación de terminal*.

Como su nombre sugiere, la emulación de terminal es simplemente un método para persuadir a un micro de que actúe como un terminal dado. De forma muy simple, el software para emulación de terminal traduce los caracteres de control de terminal entrantes (tales como los que se utilizan para limpiar la pantalla o posicionar el cursor) en instrucciones comprensibles para el ordenador que se está empleando. Del mismo modo, si el anfitrión espera del terminal una secuencia de caracteres de control, el software de emulación se la proporcionará.

Casi todos los sistemas comprenderán un subjuego de caracteres de control ASCII. Algunos de los caracteres más útiles son Control-S (ASCII 19, conocido como "XOFF"), que detiene temporalmente la recepción de datos; Control-Q (ASCII 17, conocido como "XON"), que la restablece; Control-J (ASCII,10, conocido como "LF"), que fuerza un salto de línea sin un retorno de carro, y Control-G (ASCII 7, conocido como "BEL"), que hace sonar la campanilla de la consola. El último carácter puede ser útil si se necesita atraer la atención del operador del sistema del terminal receptor.



Kevin Jones

El protocolo XModem

Habiendo visto detalladamente cómo se transmiten los datos ASCII, consideremos ahora cómo se comunica el software. La transmisión de software escrito en BASIC es directa. La mayoría de los micros soportan algún medio de convertirlos en su formato de programa comprimido a forma ASCII; en el BBC utilizamos *SPOOL; en el Commodore 64, LISTamos un archivo en disco o cinta; en el Tandy se emplea la instrucción CSAVE<nombrearchivo>,A; etc. El archivo así decodificado se transmite después y es cambiado nuevamente de formato en el otro extremo.

Sin embargo, los archivos CP/M de instrucciones (.COM) no se pueden convertir a forma ASCII, y cualquier intento por transmitir uno resulta un fracaso. Por este motivo se desarrolló un protocolo denominado *XModem*. El XModem, que es una característica de algunos software de comunicaciones, simplemente lee un archivo CP/M del disco y lo transmite en un formato binario estándar. Por supuesto, el terminal receptor debe soportar XModem.

En un futuro capítulo estudiaremos con mayor detalle el empleo de las comunicaciones por ordenador. De momento veamos someramente la gama de actividades en que se puede participar al disponer de un modem.

Correo electrónico es el nombre que identifica a los sistemas en los que los usuarios pueden intercambiar mensajes privados mediante la transmisión de los mismos a un ordenador central, donde se los almacena hasta que el destinario conecta con el sistema y los recupera.

A la mayor parte de los usuarios de micros personales les interesan dos aplicaciones determinadas. La primera es el intercambio de software a través del teléfono. Como ya hemos visto, transmitir programas en BASIC es muy simple y resulta más rápido, más sencillo y más económico que enviar cassettes por correo. Si está escribiendo un programa que le plantea dificultades, puede transmitirle una copia del mismo a sus amigos para ver si pueden ayudarlo. De ser así, ¡ellos le vuelven a transmitir a usted una versión operativa!

El segundo uso de los aficionados son los tablones de anuncios. Éstos permiten que usted le pase información a otros usuarios, deje notas solicitando ayuda técnica, gaste bromas, cargue software de dominio público, practique juegos, etc.

Los tablones de anuncios los llevan aficionados y normalmente no cobran por los mismos ninguna tarifa (si bien algunos tablones pueden cobrarle una tarifa nominal inicial, de alrededor de 250 ptas., con el objeto de cubrir los costos de funcionamiento). Más adelante analizaremos en profundidad los tablones de anuncios. En el próximo capítulo, no obstante, veremos cómo seleccionar modems y software para comunicaciones.

Muchos motores

Le enseñamos cómo controlar varios servomotores simultáneamente, conectándolos al ordenador a través de la puerta para el usuario

Hay ocho líneas de datos que se pueden conectar a motores, si bien la caja buffer que hemos diseñado sólo puede utilizar cuatro. Se podrían emplear las ocho líneas duplicando el sistema de circuito de la caja de salida para las otras cuatro.

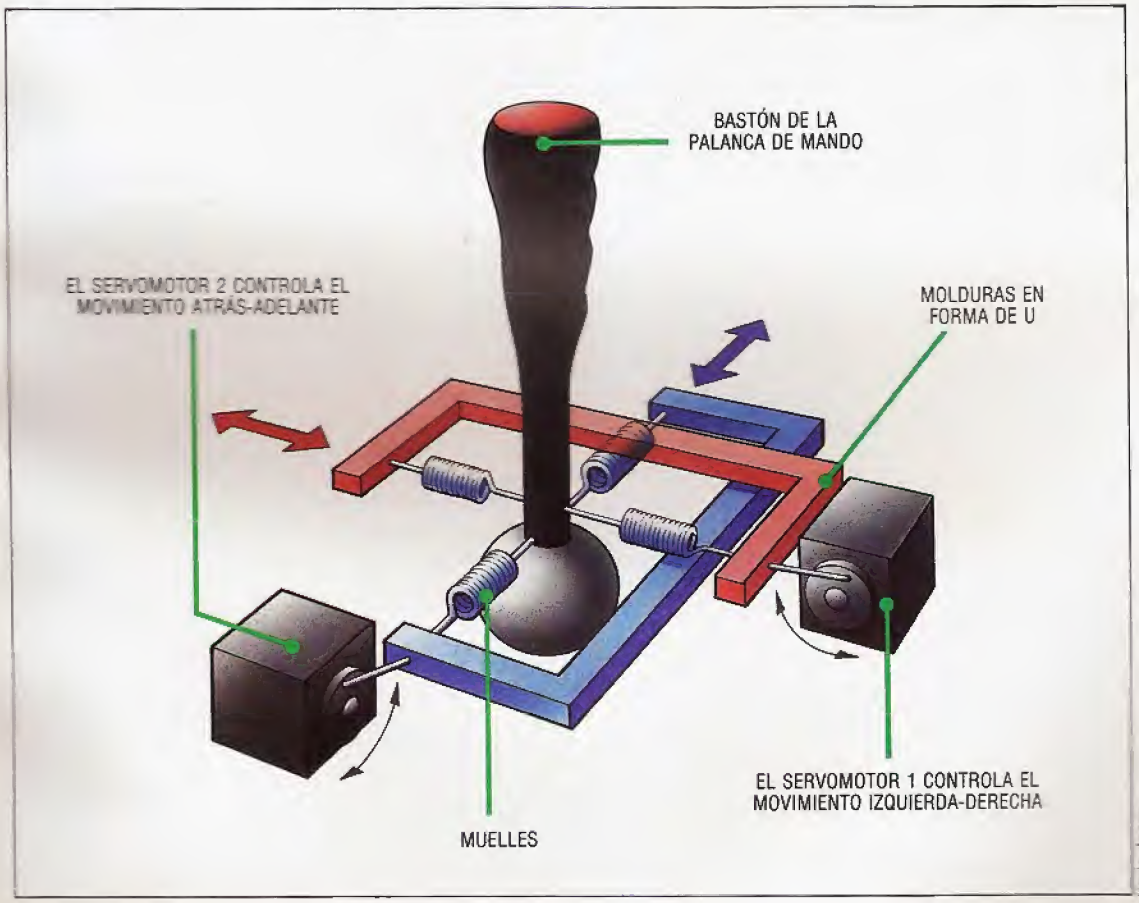
Para controlar simultáneamente ocho motores debe modificarse ligeramente el algoritmo para un solo motor que hemos desarrollado. Los impulsos son iniciados todos juntos, pero el segundo bucle de espera se reemplaza por una tabla de referencia. Se reservan 255 posiciones de memoria para la tabla y se establecen inicialmente en 255 ($FF).

Luego se entran en esta tabla de excepciones (cuando hay un motor apagado). Por ejemplo, si la línea de datos 2 se ha de apagar tras una cuenta de 20, se alterará la vigésima entrada de la tabla, de 11111111 binario ($FF) a 11111011 ($FB). Observe que en el listado en assembly esto se efectúa mediante un AND con el valor que ya esté en la tabla. Después de entradas todas las excepciones en la tabla, se inicia el bucle de espera, pero esta vez cada elemento de la tabla se opera mediante AND con la puerta para el usuario.

El algoritmo para controlar motores múltiples es:

1) Especificar el ángulo de cada motor almacenando los ángulos en ocho bytes (de ANGULO+0 a ANGULO+7).
2) Establecer altos todos los bits de datos de la puerta para el usuario, para iniciar todos los impulsos simultáneamente.
3) Insertar las excepciones en la tabla de referencia.
4) Esperar durante un milisegundo.
5) Cargar en el acumulador el número binario 11111111 ($FF). Luego operar el acumulador mediante AND con cada elemento de la tabla de referencia, de uno en uno. Al encontrar una excepción, se apagará el bit apropiado. Dado que la operación mediante AND continúa hasta el final de la tabla, este bit permanecerá apagado hasta el final.
6) Volver a establecer las excepciones de la tabla otra vez al binario 11111111, a punto para el próximo impulso.

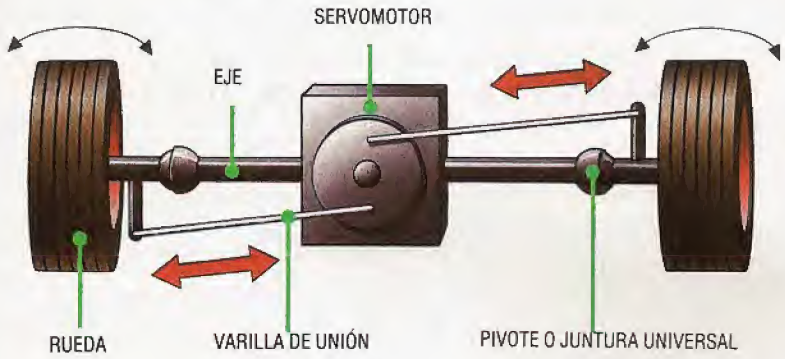
Los listados ofrecidos para el BBC Micro poseen la misma rutina inicial (líneas 10 a 280) en ambos programas. El primer listado es para el control de un solo servomotor conectado a una de las líneas de la puerta para el usuario. El temporizador de eventos



Kevin Jones

Realimentación por palanca de mando
Las palancas de mando se emplean normalmente para proporcionar información direccional a utilizar por el software. Una posible aplicación para ordenador de los servomotores consiste en emplearlos para permitir que el software controlador realimente con su información el bastón de la palanca de mando. Se utilizan dos servomotores para empujar y tirar del bastón en cada plano horizontal, proporcionando importantes datos de realimentación en los simuladores de vuelo más perfeccionados. Una palanca de mando que "se oponga" bajo la mano en respuesta a un movimiento de control por parte del piloto, mejora la simulación al proporcionar al usuario una información táctil además de visual

Mecanismo de dirección



En nuestro robot el control direccional se obtiene mediante el control independiente de dos motores paso a paso bidireccionales. Una posible alternativa a esta disposición sería tener un motor paso a paso para activar el vehículo y un servomotor para dirigirlo. El diagrama muestra un servomotor montado sobre el eje de las ruedas, empujando o tirando de las varillas que conectan las ruedas para dirigir el vehículo. Otra disposición podría ser utilizar un mecanismo de dirección de cremallera y piñón, montando el engranaje del piñón en el husillo del servomotor

se prepara mediante BASIC. Un procedimiento de inicialización ensambla la rutina manejadora de eventos antes de que el programa principal la ejecute permitiendo el evento 5.

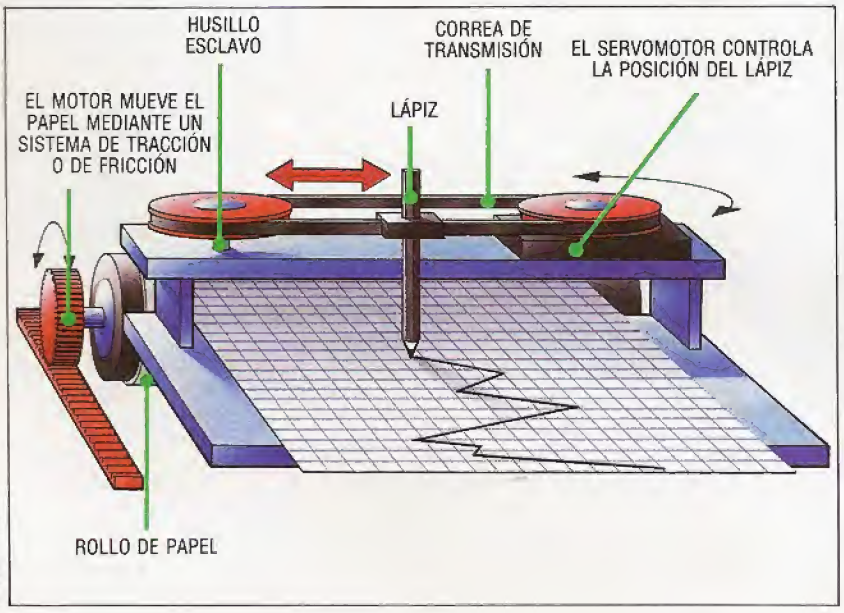
En el BBC Micro, el sistema operativo incluye un temporizador para el usuario en centésimas de segundos. Estableciéndolo en 2 centisegundos (dos interrupciones de 10 milisegundos cada una) y utilizando luego el vector de "eventos", el procesador saltará al código "impulsador" en el momento adecuado. Puesto que el sistema operativo se diseñó para emplear los eventos, el programa sólo ha de retornar (RTS) desde la subrutina, y no usar RTI.

El segundo listado, para el control de múltiples servomotores, utiliza primero un bucle en BASIC para inicializar la tabla de referencia con valores $FF. Si cada elemento de la tabla saliera de uno en uno hacia la puerta para el usuario, todos los impulsos continuarían durante dos milisegundos. Sin embargo, se pueden hacer excepciones y apagar cada motor por turno. Las excepciones se insertan en la tabla empezando por el motor 7, mediante el empleo de un desplazamiento (en el registro X) proporcional a la longitud del impulso. Luego la tabla sale hacia la puerta para el usuario mediante el direccionamiento indirecto de cada elemento de uno en uno, esta vez utilizando el direccionamiento indirecto postindexado (donde el procesador le suma el valor del registro Y a la dirección hallada en un vector de página cero).

Los patrones de bits (excepciones) que deben enviarse a la puerta para el usuario para controlar los motores se producen del siguiente modo. Se requiere el binario 01111111 para apagar el motor 7; el binario 10111111 para el motor 6; el 11011111 para el motor 5, etc. Éstos se generan cargando el registro A con $FF y limpiando el flag o indicador de arrastre. Luego, como la rutina trata cada excepción de una en una, estos bits se desplazan una posición hacia la derecha. En el primer desplazamiento, el bit de arrastre se desplaza al bit 7, el bit 7 se desplaza al bit 6, y así sucesivamente, con el bit 0 reemplazando al bit de arrastre. El patrón de bits requerido para apagar cada motor se guarda temporalmente en la pila. Cada entrada de la tabla se opera mediante AND con la entrada anterior (cuando ésta sale) para asegurar que una vez apagado un impulso éste permanezca apagado.

Habiendo generado el código máquina, el evento 5 se limpia para la acción. Por consiguiente, pulsando la tecla Shift junto con una tecla numérica del 1 al 8 se selecciona uno de los motores, mientras que pulsando las teclas del 1 al 9 los motores se colocan en posición.

Trazador de gráficos
Se puede diseñar un trazador de gráficos utilizando un servomotor para mover el lápiz hacia adelante y hacia atrás, mientras un motor paso a paso va moviendo el papel por debajo de aquél. El movimiento angular del husillo del servomotor se traduce en el movimiento lineal hacia adelante y atrás del lápiz gracias a una correa de transmisión. El movimiento del lápiz se corresponde con los cambios de una variable del eje vertical (p. ej., temperatura o presión barométrica); la alimentación continua del papel depende a menudo del transcurso del tiempo



Kevin Jones

Utilización de los listados

Para usar los listados para el BBC, simplemente éntrelos, guárdelos y luego ejecútelos. Tanto el listado de un solo servomotor como el de múltiples servomotores se ejecutan con la rutina inicial común (de la línea 10 a la 750).

Para el Commodore 64, el segundo algoritmo utiliza las mismas teclas de instrucciones que el listado para el BBC. El programa de llamada en BASIC permite establecer de forma independiente la posición de cada motor: la tecla Shift y una tecla numérica del 1 al 8 seleccionan el motor deseado, y una tecla del 1 al 9 define la posición requerida.

Si posee un ensamblador, entonces digite el listado fuente y ensámblelo en un archivo objeto que subsiguientemente se pueda cargar mediante el programa de llamada en BASIC. De no ser así, digite el cargador en BASIC para el lenguaje máquina y ejecútelo. Digite NEW antes de cargar y ejecutar el programa de llamada en BASIC. Si utiliza el cargador en BASIC, puede omitir las líneas 30 y 40.

Commodore 64: control de servomotores múltiples

Código fuente

```
1000 ;++++++++++++++++++
1010 ;++++++++++++++++++
1020 ;++                ++
1030 ;++  CONTROL MULTIPLES  ++
1040 ;++  SERVOMOTORES CBM   ++
1050 ;++                ++
1060 ;++++++++++++++++++
1070 ;++++++++++++++++++
1080 ;
1090 PUERTA=56577    ;REGISTRO DATOS PUERTA USUARIO
1100 ANGULO=1228     ;POSICION VALOR ANGULO
1110 CPAG=$FB        ;PUNTERO PAGINA 0 A TABLA

1120 ;
1130 *=$0334
1140 ;
1150        SEI          ; INTERRUPCIONES DESACTIVADAS
1160        LDA $0314    ; VECTOR IRQ EXISTENTE
1170        LDX $0315
1180        STA $03C4
1190        STX $0314
1200        LDA $0315
1210        LDX $03C5
1220        STA $03C5
1230        STX $0315
1240 ;
1250 ;++++ INICIALIZAR TABLA ++++
1260 ;
1270        LDA #$FF
1280        LDY #$00
1290 TABLA
1300        STA (CPAG),Y
1310        DEY
1320        BNE TABLA
1330        CLI          ; INTERRUPCIONES ACTIVADAS
1340        RTS
1350 ;
1360 ;++++ MANEJADOR EVENTOS ++++
1370 ;
1380        PHP
1390        PHA          ; GUARDAR REGISTROS
1400        TYA          ; EN LA PILA
1410        PHA
1420        TXA
1430        PHA
1440 ;
1450 ;++ EMPEZAR IMPULSO, PARA ALGUNOS MOTORES
1460 ; SE PODRIA COMENZAR ANTES DE LLENAR LA
1470 ;TABLA Y REDUCIR ASI EL BUCLE DE ESPERA DE ABAJO ++
1480 ;
1490        LDA #$FF
1500        STA PUERTA
1510 ;++ LLENAR TABLA CON EXCEPCIONES ++
1520        LDX #$07
1530        LDA #$FF
1540        CLC
1550 EXCEPT
1560        ROR A
1570        PHA          ; PATRON DE BITS
1580        LDY ANGULO,X ; TOMAR DESPLAZ. MOTOR X
1590        AND (CPAG),Y ; CONSERVAR PATRON EXISTENTE
1600        STA (CPAG),Y ; PERO MODIFICADO PARA MOTOR X
1610        PLA
1620        DEX
1630        BPL EXCEPT
1640 ;++ AHORA LA TABLA ESTA CARGADA ++
1650        LDY #$30
1660 ESPERA
1670        DEY          ; DEJAR PASAR
1680        BNE ESPERA   ; UN POCO DE TIEMPO
1690 ;
1700        LDA #$FF     ; TODOS LOS IMPULSOS ENCEN-
1710        LDY #$00     DIDOS
1720 BUCLE
1730        AND (CPAG),Y ; PERO OPERAR CON MASCARA
1740        STA PUERTA   CADA ELEMENTO ; DE LA TABLA
1750        INY          POR TURNO
1760        BNE BUCLE
1770 ;
1780        LDX #$07
1790        LDA #$FF
1800 LIMPIA
1810        LDY ANGULO,X ; BORRAR TODAS LAS EXCEPCIONES
1820        STA (CPAG),Y
1830        DEX
1840        BPL LIMPIA
1850 ;++ AHORA SE DEBEN TERMINAR TODOS LOS IMPULSOS ++
1860        PLA
1870        TAX
1880        PLA          ; RESTAURAR REGISTROS
1890        TAY
1900        PLA
1910        PLP
1920        JMP $EA31
```

Programa cargador en BASIC

```
10 REM ****   CARGADOR EN BASIC PARA   ****
20 REM ****   MULTIPLES SERVOMOTORES   ****
30 :
40 FOR I=820 TO 922
50 READ A: POKE I,A
60 CC=CC+A
70 NEXT I
80 READ CS:IF CS<>CC THEN
   PRINT"ERROR EN SUMA DE
   CONTROL":STOP
100 DATA120,173,20,3,174,196,3,141,196
110 DATA3,142,20,3,173,21,3,174,197,3
120 DATA141,197,3,142,21,3,169,255,160
130 DATA0,145,251,136,208,251,88,96,8
140 DATA72,152,72,138,72,169,255,141,1
150 DATA221,162,7,169,255,24,106,72
160 DATA188,0,48,49,251,145,251,104
170 DATA202,16,243,160,48,136,208,253
180 DATA169,255,160,0,48,251,141,1,221
190 DATA200,208,248,162,7,168,255,188
200 DATA0,48,145,251,202,16,248,104
210 DATA170,104,168,104,40,76,49,234
220 DATA13072:REM"SUMA DE CONTROL"
```

Programa de llamada en BASIC

```
10 REM ****   MULTIPLES SERVOMOTORES   ****
20 :
30 DN=8:REM SI CASSETTE ENTONCES
   DN=1
40 IF A=0 THEN A=1:LOAD"MULTISER V.
   HEX",DN,1
50 POKE 778,88: POKE 779,3:REM
   PUNTERO A MANEJADOR EVENTOS
60 POKE 251,0: POKE 252,49:REM
   ESTABLECER PUNTERO PAGINA CERO
70 RDD=56579: POKE RDD,255:REM
   TODAS SALIDA
80 MC=820: SYS MC
90 :
100 GET K$:IF K$="" THEN 100:REM
    ESPERAR TECLA
110 REM ** ALTERAR POSICION MOTOR **
115 AK=ASC(K$)
120 IF AK>48 AND AK<58 THEN POKE
    12288+SERVO,VAL(K$)*20
130 IF AK>32 AND AK<40 THEN
    SERVO=ASC(K$)-35
140 IF K$<>"E" THEN 100:REM "E" PARA
    SALIR
```

Control de un solo servomotor

```
  755 REM ************************************
  756 REM *        Ensamblar el codigo maquina
  757 REM ************************************
10000 DEF PROCinicial
10010 DIM espacio% 200
10020 FOR C=0 TO 2 STEP 2
10030   puertab=&FE60
10040   P%=espacio%
10050   angulo=P%
10060   P%=P%+1
10070   [OPT C
10080   .manejador eventos
10090   \guardar registros primero
10100   PHP:PHA:TYA:PHA:TXA:PHA
10110   LDA &&04
10120   LDX &xreloj
10130   LDY &yreloj
10140   JSR &FFF1 reinicializar reloj
10150   LDA &&FF
10160   STA puertab
10170   \esperar aprox. 1mseg
10180   LDY &&FF
10190   .BUCLE DEY
10200           BNE BUCLE
10210   \y contar impulso
10220   LDY angulo
10230   .BUCLE1 DEY
10240           BNE BUCLE1
10250   \detener todos los impulsos de salida
10260   LDA &&0
10270   STA puertab
10280   PLA:TAX:PLA:TAY:PLA:PLP
10290   RTS
10300   ]
10310   NEXT C
10320 REM apuntar al manejador eventos
10330 !&220=manejador eventos OR (!&220 AND &FFFF0000)
10340 ENDPROC
```

Listados para el BBC Micro

Rutina inicial común

```
 10 MODE 0
 15 REM ************************************
 16 REM          Preparar el temporizador etc.
 17 REM ************************************
 20 osbyte=&FFF4
 30 A%=&97 :X%=&62 :Y%=&FF
 40 CALL osbyte: REM preparar puerta B para salida
 50 CLS
 60 DIM p%(8)
 70 DIM reloj% 12, leer% 12
 80 xreloj=reloj% MOD 256
 90 yreloj=reloj% DIV 256
100 xleer=leer% MOD 256
110 yleer=leer% DIV 256
120 PROCinicial
130 FOR i%=angulo TO angulo+8:angulo?i%=128:NEXT
140 t=.02:REM seg entre impulsos
150 hora%=&FFFFFFFF-(t*100)+1
160 reloj%?4=&FF      :REM cargar byte mas alto
170 !reloj%=hora%     :REM establecer reloj, permitir eventos
180 +FX14,5
190 A%=4:X%=xreloj   :Y%=yreloj   :CALL &FFF1
195 REM ************************************
196 REM          El controlador BASIC
197 REM ************************************
200 CLS
210 PRINT"PULSAR SHIFT+NUMERO para seleccionar motor"
220 PRINT"PULSAR NUMERO para seleccionar angulo"
225 motor=1
230 REPEAT
240   A=GET
250   IF A>&2F AND A<&3A THEN
      a=(A-&30)*10*(225/90):angulo?(motor-1)=a:PRINTTAB
      (10,motor);"motor ";motor;"motor "angulo "angulo "a÷(90/255)
260   IF A>&20 AND A<&2A THEN motor=A-&20
270   UNTIL 0
280 END
290
```

Control de múltiples servomotores

```
 755 REM ************************************
 756 REM          Ensamblar el codigo maquina
 757 REM ************************************
 760 DEF PROCinicial
 770 DIM espacio% 600
 780 FOR C=0 TO 3 STEP 3
 790   paginacero=&70     :REM libre para usuarios
 800   puertab=&FE60 :osword=&FFF1
 810   P%=espacio%
 820   angulo=P% :P%=P%+8   :REM potencialmente 8 motores
 830   tabla=P% :P%=P%+256 :REM 256 longitudes de impulso
       posibles
 835   FORI%=tabla TO tabla+&100:?100:?I%=&FF:NEXT
 840   lowtabla%=tabla MOD 256
 850   hightabla%=tabla DIV 256
 860   ?paginacero=lowtabla% :paginacero?1=hightabla%
 870   [OPT C
 880   .manejador eventos
 890   PHP:PHA:TYA:PHA:TXA:PHA
 900   LDA &&04
 910   LDX &xreloj
 920   LDY &yreloj
 930   JSR osword
 990     /Comenzar impulso, para algunos motores seria posible
         comenzar de abajo
1000     /antes de llenar la tabla y reducir asi el bucle de esperar de abajo
1010   LDA &&FF :STA puertab
1020     /llenar tabla con excepciones
1030   LDX &&7 :LDA &&FF :CLC /preparar patron bits
1040   .excepciones
1050     ROR A :PHA /patron de bits
1060     LDY angulo,X /tomar desplazamiento correspondiente al angulo
         del motor X
1070     AND (paginacero),Y /conservar patron bits existente
1080     STA (paginacero),Y /pero modificado para motor X
1090     PLA :DEX
1100     BPL excepciones
1110   /ahora la tabla esta cargada, dejar pasar un poco de tiempo
1120   LDY &&60
1130   .esperar DEY
1140           BNE esperar
1150   LDA &&FF /todos los impulsos encendidos
1160   LDY &&0
1170   .bucle AND (paginacero), Y /pero enmascarar cada elemento
1180           STA puertab /de la tabla por turno
1190           INY
1200           BNE bucle
1201   LDX &&7 :LDA &&FF
1203   .limpiar
1205     LDY angulo,X /volver a borrar todas las excepciones
1207     STA (paginacero),Y
1208   DEX
1209   BPL limpiar
1210     /ahora todos los impulsos deben haber terminado
1220   PLA:TAX:PLA:TAY:PLA:PLP
1230   RTS
1240   ]
1250   NEXT C
1260 !&220=manejador eventos OR (!&220 AND &FFFF0000)
1270 ENDPROC
```

Aprender jugando

Esta vez revisaremos diversos programas educativos destinados a los niños más pequeños

Story machine (La máquina de las historias), producido por Spinnaker, está diseñado para niños de cinco a nueve años de edad. A través de él se pretende enseñar las reglas de la sintaxis, mejorar la ortografía y estimular la escritura expresiva.

El programa contiene un diccionario de 52 palabras, de las cuales cinco son nombres propios: los nombres de un niño, una niña, un gato, un perro y una criatura denominada *bumpus*, todos los cuales deben ser especificados por el usuario al comienzo del programa.

A partir de este diccionario el niño construye frases simples que luego son representadas por los personajes en la pantalla. Por ejemplo, la frase "Mauricio besa flores" visualizará un personaje que represente a un niño (al cual el usuario habrá llamado Mauricio), unos caracteres con forma de flores y, cuando éstos se acerquen lo suficiente, aparecerán corazones intermitentes indicando los besos.

Si una palabra está mal escrita o se la emplea de

Story machine (La máquina de las historias)

forma incorrecta, el ordenador se negará a aceptarla, visualizando un mensaje en que explica lo que está mal y en que pide que la palabra se modifique o se sustituya. Al final de cada frase (es decir, después de cada punto) el programa intentará ponerla en escena. Cuando se ha construido una historia, hay una opción para representarla completa.

Si el usuario se aburre de escribir sus propias historias, *Story machine* ofrece otras dos opciones: puede compartir la escritura de la historia con el programa (turnándose para añadir palabras) o dejar que el ordenador haga todo el trabajo y que escriba él la historia. Esto es factible porque el diccionario está dividido por elementos de la frase (sustantivos, verbos, preposiciones, etc.) y posee un algoritmo gramatical estricto para generar sus propias frases.

La mayoría de las palabras del diccionario, así como las estructuras oracionales utilizadas por el programa, son muy sencillas. Los verbos, en particular, describen acciones que se pueden describir fácilmente, tales como "salta" y "va". Es interesante el hecho de que se haya incluido un verbo imaginario, *zot*, el cual parece significar "pegar" o "golpear". De ser así, ello estaría en desacuerdo con la regla mantenida por Spinnaker, de que está mal herir a otras personas o a animales, y por lo tanto el programa no admitirá ninguna acción que implique un daño físico a otros.

El programa posee dos reglas gramaticales fundamentales: una oración debe contener un sujeto, un verbo y un objeto, en el orden correcto; y las formas en plural y en singular deben ser coherentes. Más allá de estas reglas, la sintaxis es menos satisfactoria debido a las inmensas variaciones del uso del idioma inglés. Por ejemplo, el ordenador aceptará la frase *"Houses eat rocks"* (Casas comen rocas) pero no aceptará la frase *"Girls eat rocks"* (Chicas comen rocas) sin el artículo definido antes de la palabra "chicas".

El programa parece incluso quebrantar sus propias reglas. Después de la frase *"Bumpuses eat rocks"* (Bumpuses comen rocas), el programa generó la oración *"They walk to its fence"* (caminan hacia su verja). Normalmente, al encontrarse con la palabra *"its"* (su), el ordenador generaría el mensaje *"Whose?"* (¿De quién?) y solicitaría que se cambiase la palabra. El programa también ha hecho caso omiso de su regla de no mezclar formas en singular y plural. Éstos se podrían considerar errores menores, pero un programa que se vende como educativo debería al menos ser coherente y *debe* ser siempre correcto.

Kids on keys (Niños a las teclas) es otro producto de Spinnaker, dirigido a niños de tres a nueve años. El programa tiene como objetivo enseñar al usuario a identificar palabras, letras y números. En el paquete hay tres juegos diferentes, cada uno de los cuales posee varios niveles de dificultad. En el primer juego, una letra se desplaza hacia abajo de la pantalla y el niño debe pulsar la tecla correcta antes de que la letra llegue al límite inferior. Al cabo de 15 letras, aparece flotando hacia abajo un globo que contiene una palabra. Ésta también debe digitarse correctamente antes de que el globo llegue hasta la parte inferior de la visualización. El juego, por lo menos, consigue que el niño disfrute mientras se familiariza con el trazado del teclado.

En el segundo juego se van desplazando hacia abajo de la pantalla una serie de imágenes (sprites ampliados) y el niño debe digitar el nombre del objeto antes de que lleguen a la parte inferior. Después de esto, hay una ronda extra en la cual descienden los mismos objetos, pero en este caso faltándoles dos cuartos de la imagen, lo que hace que resulte más difícil reconocerlos. En este punto, el

Kids on keys (Niños a las teclas)

juego tiende a convertirse en una prueba de reflejos, mientras el jugador busca frenéticamente las teclas para digitar las letras correctas. Un importantísimo inconveniente del programa es la ausencia de una facilidad para borrar, con lo cual la pulsación errada de una tecla se vuelve sumamente frustrante.

Quizá el problema más serio del juego sea que algunos de los sprites están mal dibujados y es difícil identificarlos. Por ejemplo, si un niño decide que un determinado dibujo es un hombre, podría digitar "hombre" de forma correcta, sólo para descubrir que el sprite continúa su descenso porque el programa considera que es un "oso" o un "niño". El pequeño puede entonces llegar a la conclusión de que su ortografía es incorrecta, porque el programa no le explica el motivo del fallo (puesto que no posee facilidad alguna para analizarlo). Además, una vez que el sprite ha llegado a la parte inferior de la pantalla, el programa no ofrece ninguna indicación acerca de cuál era la respuesta acertada. Un buen programa educativo no sólo le proporcionaría al usuario la posibilidad de intentarlo otra vez, sino que también visualizaría la respuesta correcta después de haber obtenido varias respuestas equivocadas.

En el tercer juego se visualizan en la pantalla cinco figuras y una palabra, y el niño debe encontrar la pareja correcta para la palabra. Este juego tiene menos relación con la velocidad que los dos anteriores, si bien también adolece de la pobreza de los gráficos.

Aprender a leer

Macmillan, la editorial británica de publicaciones educativas, ha desarrollado una serie de programas para el Spectrum bajo el título colectivo de *Learn to read* (Aprender a leer). Estos cinco programas (producidos en colaboración con Sinclair Research) derivan del esquema de lectura *Gay way* de Macmillan, que ha obtenido un éxito enorme; se trata de un curso centrado en el niño que se utiliza ampliamente en las escuelas primarias. Dado que el curso *Gay way* está orientado hacia la enseñanza individual, posee la ventaja de que los usuarios aprenden a leer a su propio ritmo. Una serie de programas para ordenador es una ampliación natural de ello, puesto que utilizar un ordenador es evidentemente una actividad orientada hacia el individuo.

Los cinco paquetes componen un curso preliminar de lectura completo. El primer programa presenta seis animales (*Deb* la rata, *Sam* el zorro, etc.) y éstos se mantienen a lo largo de todo el curso para proporcionar un sentido de continuidad y familiaridad.

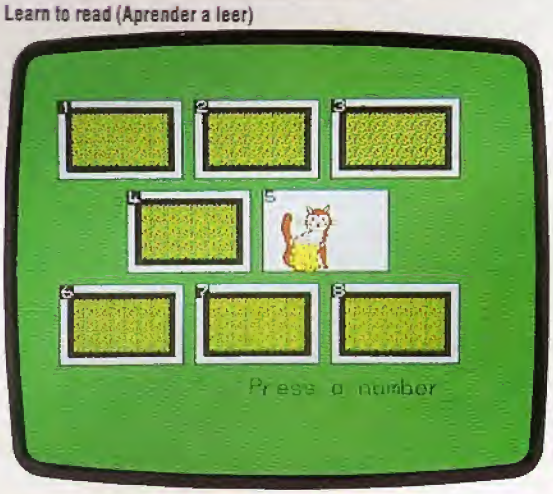
Los programas dan por sentado que el niño no posee ningún conocimiento previo de ordenadores ni de lectura. En cada programa la selección de las opciones es directa. Por ejemplo, el primer programa visualiza cinco opciones, y rodeando el nombre de cada una va apareciendo, de uno en uno, un rectángulo intermitente. El usuario espera hasta que el "cursor" rectangular rodee la opción requerida y entonces pulsa cualquier tecla. Para utilizar este menú no se necesita emplear ninguna otra instrucción.

Una vez escogida una opción, el ordenador demuestra lo que se requiere y le solicita luego al niño que dé una respuesta. Si el niño da una respuesta equivocada, se le ofrecerán varias oportunidades más antes de que el ordenador visualice la respuesta correcta.

Los seis animales que aparecen en el programa están dibujados en gráficos de alta resolución, y éstos son muchísimo más atractivos que algunos de los que se incluyen en los otros juegos educativos que hemos analizado. Desde el punto de vista del entretenimiento, quizá el valor de la serie *Learn to read* no sea tan alto como el de muchos otros paquetes, pero su valor educativo parece ser muchísimo mayor, y, a la larga, tal vez mucho más provechoso para el niño.

Learn to read (Aprender a leer)

Learn to read (Aprender a leer)



Ian McKinnell

Gran Premio

Participe en uno de los juegos recreativos que han tenido mayor éxito: "Gran Premio". Esta versión ha sido escrita para los ordenadores Atari

En la pista de carreras, cuatro coches encabezan la competición en una carrera desenfrenada; usted debe evitar tener un accidente mientras trata de recorrer la mayor distancia posible. Utilice la palanca de mando para dirigir su coche y el botón rojo para cambiar de velocidad. ¡Atención!, este programa emplea lenguaje máquina, de modo que tome precauciones antes de lanzarse a competir.

```
 0 REM ** GRAN PREMIO, ESCRITO **
 1 REM ** POR PAUL DUNNING      **
 2 DIM S(4),V$(10),E$(100)
 3 GRAPHICS 5:V$="LENT":POKE 752,1
 5 GOSUB 1000
 6 POKE 765,2:COLOR 2:PLOT 0,0:DRAWTO 79, 0:PLOT
   79,47:DRAWTO 79,30:DRAWTO 0,30:POSITION 0,47:XIO
   18,#6,0,0,"S:"
 7 POKE 765,3:COLOR 3:PLOT 79,29:DRAWTO 79,1
   :DRAWTO 0,1:POSITION 0,29:X10 18,#6,0,0,"S:"
 8 POKE 712,148
10 PP=PEEK(106)-16:POKE 54279,PP
20 PM=PP*256:POKE 559,62
21 FOR Q=53256 TO 53260:POKE Q,3:POKE
   Q-8,RND(1)*255:NEXT Q
22 POKE 53248,100:POKE 53249,100:POKE 53250,
   100:POKE 53251,50
23 POKE 704,109:POKE 705,89:POKE 706,79: POKE
   707,29:POKE 53277,3:POKE 559,62
24 FOR Q=PM+1024 TO PM+2048:POKE Q,0:NEXT Q
30 RESTORE 210:FOR Q=PM+1144 TO PM+1280:READ
   A:IF A=-1 THEN 32
31 POKE Q,A:NEXT Q
32 RESTORE 210:FOR Q=PM+1360 TO PM+1536:READ
   A:IF A=-1 THEN 34
33 POKE Q,A:NEXT Q
34 RESTORE 210:FOR Q=PM+1576 TO PM+1792:READ
   A:IF A=-1 THEN 36
35 POKE Q,A:NEXT Q
36 RESTORE 210:FOR Q=PM+1842 TO PM+2048:READ
   A+IF A=-1 THEN 40
37 POKE Q,A:NEXT Q
40 POKE 623,1
50 SOUND 0,255,12,5:SOUND 1,145,12,5:SOUND
   2,200,12,5
60 FOR Q=53256 TO 53260:POKE Q,3:POKE
   Q-8,RND(1)*255:NEXT Q
70 POKE 704,109:POKE 705,89:POKE 706,79:POKE
   707,29:POKE 53277,3:POKE 559,62
75 ST=PM+1842:VI=INT(ST/256):LT=ST-256*VI:POKE
   203,LT:POKE 204,VI:POKE 205,10
80 P1=(RND(1)*100)+100:P2=(RND(1)*100)+100:P0=
   (RND(1)*100)+100:P3=50
85 FOR Q=0 TO 3:S(Q)=(RND(1)*2)+1:NEXT Q
87 POKE 53248,P0:POKE 53249,P1:POKE 53250,P2:POKE
   53251,50
88 FOR Q=1 TO 50:X=USR(ADR(E$),7):NEXT Q
90 IF V$="RAP." THEN GOSUB 400:GOTO 120
92 POKE 53278,1
95 P0=P0+S(0):IF P0>255 THEN P0=0
98 X=USR(ADR(E$),STICK((0))
100 P1=P1+S(1):IF P1>255 THEN P1=0
105 X=USR(ADR(E$),STICK(0))
110 P2=P2+S(2):IF P2>255 THEN P2=0
120 KM=KM+0.01
124 X=USR(ADR(E$),STICK(0))
129 IF STR16(0)=0 THEN GOSUB 500
130 POKE 53248,P0:POKE 53249,P1:POKE 53250,P2
135 POKE 656,2:POKE 657,5:? "VELOCID ";V$;"    ":POKE
    656,2:POKE 657,20:? "KILMS";KM;"  "
140 CP=PEEK(53263):CS=PEEK(53255)
150 IF CP<>0 THEN 600
160 IF CS<>4 THEN 600
190 GOTO 90
210 DATA 64,228,238,238,238,238,78,68,229,245,255,
    255,233,255,255,233,255,255,245,229,68,78,238,238,
    236
215 DATA 238,238,64,-1
300 C=INT(RND(1)*3):S(C)=(RND(1)*2)+1:RETURN
400 P0=P0-(S(0)*2):IF P0<0 THEN P0=255
405 X=USR(ADR(E$),STICK(0))
410 P1=P1-(S(1)*2):IF P1<0 THEN P1=255
415 X=USR(ADR(E$),STICK(0))
420 P2=P2-(S(2)*2):IF P2<0 THEN P2=255
425 X=USR(ADR(E$),STICK(0))
430 KM=KM+0.05:RETURN
500 IF V$="RAP." THEN V$="LENT":SOUND
    0,255,12,10:SOUND 1,245,12,10:SOUND
    2,235,12,10:RETURN
510 V$="RAP.":SOUND 0,100,12,10:SOUND
    1,110,12,10:SOUND 2,90,12,10:RETURN
600 GOTO 640
605 SOUND 0,0,0,0:SOUND 1,0,0,0:SOUND 2,0,0,0
610 POKE 53248,100:POKE 53249,100:POKE 53250,100
620 TRAP 620:? "OTRA?";:INPUT V$:IF V$(1,1)="0" THEN
    KM=0:V$="LENT":? CHR$(125):GOTO 24
630 END
640 J=PEEK(203)+256*PEEK(204)
645 FOR Q=1 TO 250:IF PEEK(J+Q)=0 THEN NEXT Q
650 ST=J+Q:VI=INT(ST/256):LT=ST-256*VI:POKE 203,
    LT:POKE 204,VI
655 FOR P=1 TO 60
660 X=USR(1536):SOUND 1,PEEK(53770),120,15
670 POKE 707,PEEK(53770):NEXT P
700 POKE 53251,0:GOTO 605
1000 RESTORE 2000
1010 TRAP 1050:P=0
1020 P=P+1:READ A
1030 E$(P,P)=CHR$(A)
1040 GOTO 1020
1050 RESTORE 3000
1060 TRAP 1090:P=1536
1070 READ A:POKE P,A:P=P+1
1080 GOTO 1070
1090 TRAP 40000
1100 RETURN
2000 DATA 104,104,104,74,72,176,11,160,0,177,203,
     136,145,203,200,200,208,247,104,74,72
2010 DATA 176,11,160,0,177,203,200,145,203,136
2020 DATA 136,208,247,104,74,72,176,7,198,205,165
2030 DATA 205,141,3,208,104,74,176,7,230,205,165,205,
     141,3,208,96,END
3000 DATA 104,160,0,173,10,210,145,203,200,192,25,208,
     246,96,END
```

Llegados de Oriente

El PC-1251 y el PC-1500A, de Sharp, son dos ordenadores de bolsillo ligeros, versátiles y de precio asequible

Potencia de bolsillo
Con RAM CMOS y CPU de 8 bits, teclado QWERTY, BASIC y una gama de periféricos opcionales, estas "calculadoras" Sharp tienen el derecho de ser consideradas microordenadores a pequeña escala y auténticamente portátiles

La estrategia de marketing de la firma japonesa Sharp contrasta notablemente con la de Casio, la otra empresa fabricante de ordenadores de bolsillo. En primer lugar, produce dos máquinas bastante diferentes, mientras que Casio comercializa tres máquinas relativamente similares. Las dos máquinas de Sharp no están diseñadas para competir entre sí: el PC-1251 es el ordenador de bolsillo más pequeño que existe actualmente, y el PC-1500A es considerablemente más grande y más potente.

El Sharp PC-1251 pesa apenas 115 g y mide 135 × 70 × 10 mm. El tamaño del teclado es aun inferior a la mitad de uno estándar y la anchura de las teclas es de sólo 4 mm. Son lo suficientemente grandes, eso sí, para asegurar que, siempre que se tenga cuidado, al pulsar una tecla no se pulsen también las cuatro que la rodean. Al lado del teclado alfabético hay uno numérico con teclas más grandes, que permite utilizar el ordenador como una calculadora de bolsillo normal.

El PC-1251 posee una visualización en cristal líquido de 24 caracteres de anchura. Ésta se puede regular para diversos ángulos de visión y condiciones luminosas mediante una pequeña rueda situada en uno de los extremos del ordenador. A la derecha de la visualización hay un interruptor Mode Selector (selector de modalidad). Hay tres modalidades operativas: una permite definir la función de ciertas teclas (RSV); una segunda modalidad facilita la programación (PRO), y la tercera permite ejecutar un programa en BASIC o utilizar la máquina como calculadora (RUN). Este interruptor también se emplea para apagar la máquina.

A pesar de no responder del todo al estándar de los micros personales comunes, la versión de BASIC utilizada es buena tratándose de una máquina tan pequeña. Posee instrucciones de las que carecen algunos de los ordenadores de bolsillo de Casio, como ASC y CHR$; pero, al igual que éstos, carece de la opción ELSE para la instrucción IF...THEN. Al igual que los ordenadores Casio más pequeños, el BASIC del PC-1251 utiliza los mismos nombres de una sola letra para series y variables. Ello significa que si se empleara la variable A para retener un número, no se puede utilizar la variable en serie A$. Del mismo modo, algunas matrices podrían machacar las mismas zonas de memoria.

Esta versión de BASIC sólo produce nueve mensajes de error y todos letras únicas, lo cual carece prácticamente de utilidad para atrapar errores en los programas. Como opción adicional, la instrucción PASS permite proteger los programas mediante una palabra clave. En el BASIC del PC-1251 los números de línea están limitados a la gama del 1 al 999. En la modalidad de entrada de programas, dos teclas de cursor le permiten al programador ir desplazando las líneas del programa hacia arriba y hacia abajo. En todas las modalidades, otros dos cursores permiten el desplazamiento lateral.

Puesto que es escaso el software disponible para la máquina, la mayoría de los usuarios habrán de escribir sus programas. En este sentido, el manual resulta de ayuda en dos sentidos. En primer lugar, ofrece una guía de BASIC buena y de fácil comprensión, aunque no incluye enseñanza para principiantes. En segundo lugar, contiene los listados de nueve programas cortos, no todos los cuales son aplicaciones matemáticas (hallar raíces, desviación estándar, etc.); entre los otros programas incluidos hay uno para "practicar mecanografía" y un juego de "aterrizaje suave". Además, Sharp ofrece tres cintas que contienen una selección de programas.

Mientras que los ordenadores Casio pueden tener hasta 10 programas simultáneamente en la memoria, el PC-1251 se limitada a uno cada vez. Sin embargo, se puede utilizar un programa compuesto por varios subprogramas, separados cada uno por sentencias END. Un programa se puede de esta manera ejecutar mediante el empleo de GOTO con el número de línea apropiado. Es muy práctico tener varios programas en la memoria al mismo

tiempo, ya que no existe ninguna forma de cargar programas como no sea digitándolos cada vez que se los necesita. Además, el ordenador posee sólo 4 K de memoria, de modo que hay poco lugar para algo más que unos pocos programas modestos en BASIC. Ello significa que, casi para cualquier uso serio, son esenciales la impresora y la cassette opcionales.

Sharp PC-1500A

El otro ordenador portátil de Sharp, el PC-1500A, es un desarrollo de un modelo anterior (el PC-1500) y está diseñado con el objetivo puesto en un usuario más serio. Mide 195 × 85 × 25 mm y pesa 375 g, con lo cual su peso es tres veces mayor que el del PC-1251.

El PC-1500A posee un teclado mejor que el de su hermano y una visualización ligeramente más grande. La anchura de la LCD se amplía mínimamente (26 caracteres en lugar de 24) y en la máquina más grande ha desaparecido la facilidad del PC-1251 para regular la pantalla.

La versión de BASIC, sin embargo, constituye una gran mejora respecto a la otra. Las variables pueden tener nombres de dos letras y se puede utilizar el mismo nombre para variables en serie y numéricas sin producir ninguna confusión. Asimismo, el BASIC soporta algunas útiles facilidades para gráficos y sonido. En la LCD se pueden dibujar patrones con una resolución de 7 filas por 156 columnas, mediante el empleo de la instrucción GPRINT para definir cada una de las columnas de siete puntos. La instrucción BEEP de la máquina permite controlar el tono y la duración de las notas, que son moderadamente fuertes.

El ordenador tiene incorporados un calendario y un reloj, a los cuales se puede acceder mediante la variable TIME. El ordenador mantiene la hora aun cuando esté apagado, de modo que una vez puesto en hora se mantiene haciendo tictac. Esta facilidad sería una adición muy útil para todos los micros personales. El PC-1500A posee en su BASIC otras varias instrucciones que lo colocan en posición ventajosa respecto a muchos micros. Éstas incluyen una instrucción ON ERROR GOTO y facilidades de rastreo (TRON y TROFF). Posee la generosa cifra de 39 mensajes de error para el ordenador estándar; hay otros 16 disponibles para instrucciones utilizadas sólo con unidades accesorias. No obstante, al igual que el PC-1251, estos mensajes de error se visualizan en forma de números, por lo que podrían mejorarse con texto.

La fila superior de las teclas alfabéticas tiene programadas palabras clave del BASIC, y para identificar las 10 instrucciones se les puede instalar encima una plantilla plástica. El motivo por el cual Sharp decidió no imprimir estas instrucciones directamente en la carcasa es todo un misterio: es muy fácil que la plantilla se pierda. Las seis teclas de encima del teclado principal pueden tener programadas hasta 18 funciones.

Los usuarios del PC-1500A tendrán que escribir la mayoría de su software: son pocas las empresas que producen programas para la máquina, y la propia Sharp vende sólo una cinta de programas seleccionados. Con la máquina se suministra un libro con los listados de 53 programas, con diversas aplicaciones para cinco áreas principales: matemáticas, estadística, electricidad, oficina y juegos. Estos programas se escribieron originalmente para el PC-1500, pero todos ellos funcionan a la perfección en el PC-1500A porque la única diferencia entre los dos modelos es la cantidad de memoria RAM disponible. El modelo anterior tenía 3,5 K de memoria, mientras que el PC-1500A posee 8,5 K.

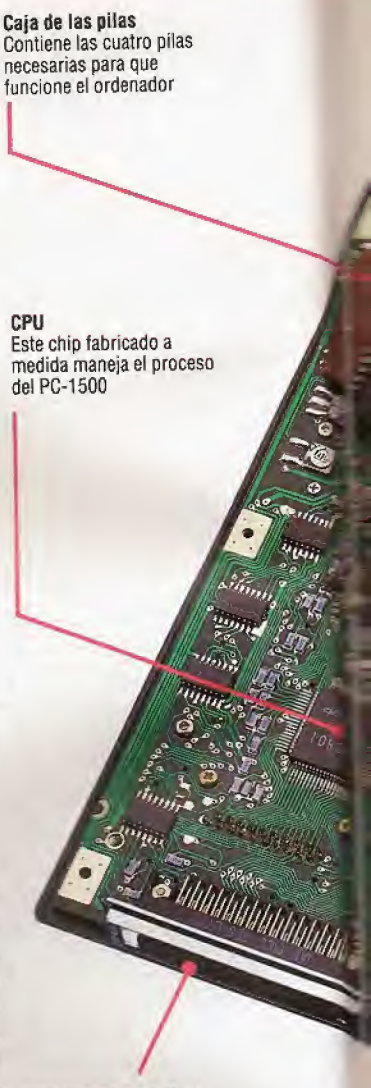
Mediante una ranura para cartuchos situada en la cara inferior de la máquina se puede incrementar la cantidad de memoria disponible. Se ofrecen cuatro cartuchos de memoria y todos ellos son bastante caros. Los paquetes de memoria estándares de 4 y 8 K conservan su contenido sólo mientras están en la máquina. Otros dos cartuchos poseen pilas, de modo que conservan su contenido aun cuando se los extraiga del ordenador. Éstos poseen una capacidad de 8 y 16 K, respectivamente.

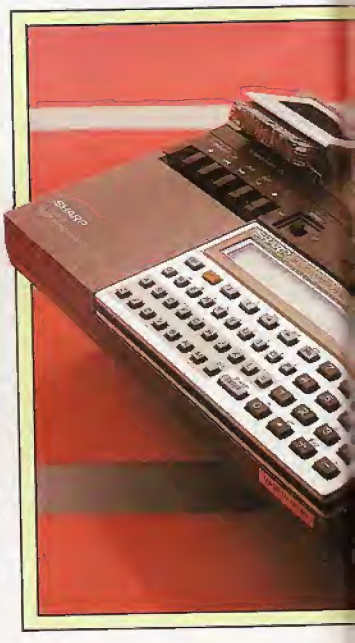
La unidad interface para impresora/plotter y cassette representa una mejor relación precio/prestaciones. La interface para cassette permite salvar y cargar programas con una grabadora de cassette común, y Sharp ofrece su propia grabadora de cassette compatible. La parte impresora/plotter de la unidad utiliza cuatro lápices de punta esférica para dibujar letras y gráficos de buena calidad a cuatro colores. Es casi idéntica a otras muchas impresoras/plotter que existen en el mercado de ordenadores personales, pero el papel que utiliza es de sólo 57 mm de ancho, lo que constituye una gran limitación.

El BASIC incluye un juego completo de instrucciones para utilizar la impresora/plotter. Éstas son: CSIZE para producir letras de diferente tamaño, ROTATE para imprimir caracteres de forma apaisada o invertida, COLOR para seleccionar el color del lápiz, LF para desplazar el papel hacia arriba y hacia abajo, LPRINT para imprimir texto, LCURSOR y GLCURSOR para desplazar el lápiz en modalidades texto y gráficos, SORGN para establecer el margen, y LINE y RLINE para dibujar una línea entre dos puntos empleando coordenadas absolutas y relativas, respectivamente. La unidad para impresora/plotter y cassette se suministra en su propia carcasa, junto con accesorios tales como un transformador de corriente (con un cable para un enchufe común) para alimentar sus pilas recargables.

Para el PC-1500A se producen otros dos accesorios importantes. Uno de ellos es una interface Centronics y RS232, que permite a la máquina comunicarse con impresoras y ordenadores de tamaño natural. Al otro periférico se lo denomina *tablero de software*. Este dispositivo es un gran teclado sensible al tacto que posee 140 teclas definibles que se pueden programar para llevar a cabo tareas utilizadas comúnmente (como calcular totales de forma automática en aplicaciones de hoja electrónica). Dado que el tablero de software es caro y para su operación requiere la interface, el precio de esta facilidad de ampliación es más bien prohibitivo.

A pesar de que el PC-1500A se puede ampliar convirtiéndolo en un sistema potente con muchas facilidades de calidad notable, hay muchos sistemas de micros de precio similar que son más versátiles y (lo más importante) están apoyados por una gama de software más amplia. Con su intento por encabezar la liga de los ordenadores de peso ligero, el PC-1500A corre el peligro de verse aventajado con holgura por muchísimos pesos medios más potentes.





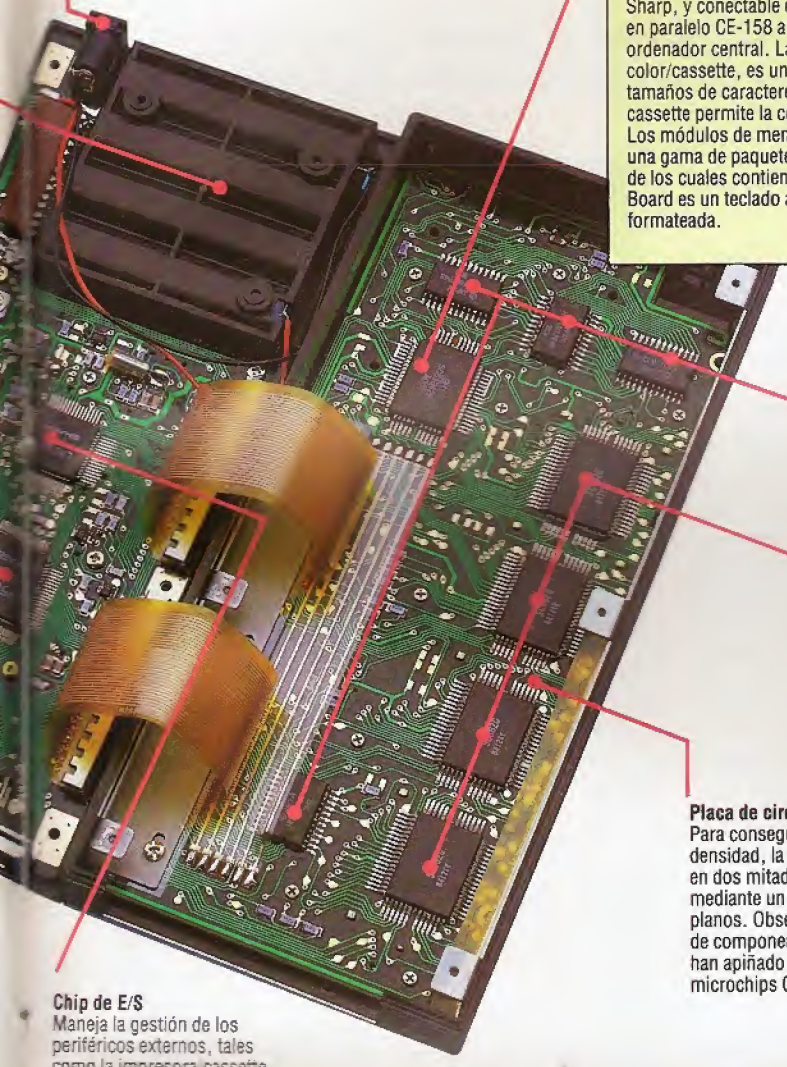
Enchufe red
Como alternativa al empleo de pilas, el ordenador puede funcionar con la corriente de la red mediante un transformador adecuado

ROM de BASIC
Este chip contiene el BASIC Microsoft que utiliza el ordenador

Extra Sharp
El Sharp PC-1251 se instala en la unidad impresora/microcassette CE-125. Ésta contiene una impresora térmica (24 caracteres por línea) y una grabadora de microcassette. La unidad mide 205 × 149 × 23 mm.
El Sharp PC-1500 es el centro de una familia ampliada de equipos Sharp, y conectable en interface a través de la interface RS232C/ en paralelo CE-158 a prácticamente cualquier sistema de micro u ordenador central. La interface CE-150, para impresora a color/cassette, es un plotter X-Y a cuatro colores con nueve tamaños de caracteres y opción para gráficos. La interface para cassette permite la conexión con dos grabadoras de cassette. Los módulos de memoria CE-151, CE-155, CE-159 y CE-161 son una gama de paquetes CMOS de RAM de entre 2 y 16 K, algunos de los cuales contienen ROM programable. El CE-153 Software Board es un teclado al tacto de 140 teclas para entrada formateada.

SHARP PC-1500A

DIMENSIONES
195 × 85 × 25 mm
PESO
375 g
MEMORIA
8,5 K de RAM
PANTALLA
LCD de 1 × 26 caracteres
OBSERVACIONES
Calendario y reloj incorporados; buen BASIC; memoria ampliable; enorme gama de periféricos



Chips de RAM
Proporcionan los 8,5 K de memoria, de los cuales hay 6,6 K disponibles para el usuario

Chips de visualización
Estos cuatro chips tratan la visualización en la pantalla LCD

Placa de circuito impreso
Para conseguir la máxima densidad, la placa se ha dividido en dos mitades, conectadas mediante un par de cables planos. Observe la escasez de componentes: éstos se han apiñado en los microchips CMOS

Chip de E/S
Maneja la gestión de los periféricos externos, tales como la impresora cassette

SHARP PC-1251

DIMENSIONES
135 × 70 × 10 mm
PESO
115 g
MEMORIA
4 K de RAM
PANTALLA
LCD de 1 × 24 caracteres
OBSERVACIONES
Tres modalidades de funcionamiento; 18 teclas definibles por el usuario; dispone de cassette/impresora

Sharp PC-1251

Sharp considera esta calculadora como su sistema de gestión de uso diario (probablemente más para el ingeniero y el científico que para el administrador); las diminutas teclas de letras de su teclado QWERTY hacen que la entrada de textos resulte difícil y cansada

Sharp PC-1500A

Este es un ordenador de bolsillo extraordinariamente flexible y del tamaño de una calculadora; sus argumentos de venta son su potencia y su equipo opcional, y podría en efecto contribuir en gran medida a aliviar la cuota de trabajo de oficina. Sin embargo, es probable que se lo considere sólo como un juguete para ejecutivos y una calculadora notable

Chris Stevens

Dibujo de imágenes

Esta vez diseñaremos visualizaciones de pantalla para el Spectrum con el fin de ilustrar dos escenarios de "Digitaya"

El diseño de la pantalla ALU implica el desplazamiento de las letras A, L y U hasta el centro de la pantalla utilizando gráficos en alta resolución. En el BBC Micro, este desplazamiento se efectuaba dibujando la letra desde un punto de partida especificado empleando instrucciones para dibujo relativo, borrándolas luego, desplazando el punto de partida y repitiendo todo el procedimiento. La misma idea se puede aplicar en la versión para el Spectrum.

La instrucción DRAW del Spectrum permite sólo el dibujo relativo (es decir, partiendo desde el último punto especificado), pero es ideal para esta particular aplicación de desplazamiento. Trazando (PLOT) un punto de partida inicial y llevando luego a cabo una serie de instrucciones DRAW para crear la forma de la letra, podemos desplazar fácilmente todo el diseño completo de la letra alrededor de la pantalla, con sólo cambiar las coordenadas del punto trazado inicialmente. El borrado se puede obtener dibujando la misma forma en la misma posición, pero con todos los colores invertidos. Este efecto se activa mediante INVERSE 1 y se desactiva mediante INVERSE 0. Por lo tanto, para cada posición que ocupe la letra la dibujaremos dos veces: una vez con INVERSE 0, para hacer aparecer la forma, y otra vez con INVERSE 1 para borrarla.

Si tomamos el ejemplo de la letra A, que se desplaza desde la izquierda, podemos colocar todas estas instrucciones dentro de un bucle FOR...NEXT. Este bucle incrementa el valor de la coordenada x del punto trazado inicialmente para la forma. Anidada en el interior de este bucle hay una segunda estructura FOR...NEXT que simplemente lleva a cabo dos veces las instrucciones de dibujo. El último valor de x es 55, que denota la posición de descanso final de la letra en la pantalla. Obviamente, no deseamos borrar la versión final de la letra, de modo que se inserta una condición para asegurar que la letra se borre (cambiando a INVERSE 1) sólo si la coordenada x es menor que 55. Los principios analizados aquí también se aplican a las otras dos letras para hacer que la L se desplace hacia arriba de la pantalla y la U lo haga desde la derecha.

Diseño esquemático de la ALU

Cuando se diseña una pantalla gráfica es importante hacer un esquema del diseño sobre papel y realizar un cálculo inicial de los valores de las coordenadas que tendrá cada forma en la pantalla. Además, también se han de situar todas las letras a imprimir en la pantalla, en términos de filas y columnas. La instantánea de la pantalla nos muestra tal diseño,

Ian McKinnell

con las dimensiones de ésta en unidades de gráficos y de caracteres.

Las palabras AND, OR y NOT se visualizan en la pantalla utilizando la instrucción PRINT AT r,c: siendo r el número de filas desde la parte superior de la pantalla, e indicando c el número de columnas desde el margen izquierdo. Los botones se dibujan empleando CIRCLE x,y,r, donde se especifican las coordenadas del centro y la longitud del radio.

Completadas las rutinas de dibujo, el programa espera a que se pulse una tecla antes de restablecer INK y PAPER a los colores originales y limpiar la pantalla. Después retorna a la rutina ALU principal. La pulsación de tecla se trata mediante INKEY$; de no pulsarse ninguna tecla, entonces se repite la condición. Para llamar a esta subrutina se debe insertar esta línea en el programa *Digitaya*:

```
4565 GOSUB 7000:REM S/R IMAGEN ALU
```

La pantalla de la puerta para la palanca de mando está diseñada para disparar rayos láser desde el centro de un conector para palanca de mando. Las patillas del conector son caracteres de punto impresos en la pantalla y el contorno tipo D se dibuja empleando gráficos en alta resolución. Para conferirle a la imagen una sensación de profundidad, se dibuja en el primer plano una serie de líneas escalonadas. El punto de partida de cada línea se halla en la línea del horizonte y se selecciona mediante una instrucción PLOT. El final de cada línea está en la parte inferior de la pantalla. Las líneas están separadas por intervalos de una unidad en el horizonte, ampliándose la separación a siete unidades en la parte inferior de la pantalla.

El hecho de que la instrucción DRAW del Spectrum sea relativa hace que la rutina sea ligeramente más complicada que si pudiéramos especificar un

punto final absoluto. Si la coordenada x del punto de partida de la línea situada más a la izquierda es 111, entonces debemos efectuar un cálculo basado en ello para hallar el desplazamiento relativo hasta el punto final. El bucle FOR...NEXT de las líneas 8030-8060 muestra este cálculo.

Como antes, es útil esquematizar las dimensiones y las coordenadas del diseño sobre el papel antes de empezar a escribir el código. La instantánea de la pantalla nos muestra tal diseño:

Los rayos láser se dibujan desde la puerta para la palanca de mando mediante el desplazamiento hasta un punto del centro de la puerta y dibujando (DRAW) luego una línea hacia un punto del horizonte escogido al azar, utilizando un color INK seleccionado al azar. Repitiendo el procedimiento con INVERSE 1, podemos borrar la línea, hacer que el haz aparezca sólo durante un breve intervalo (creando un efecto de relámpago). Sin embargo, al borrar la línea surge un problema. Dado que el rayo se dibuja desde un punto situado en el centro de la puerta, atraviesa los gráficos dibujados previamente y que representan a la puerta propiamente dicha. Cuando se borra la línea aparecen agujeros en el gráfico de la puerta, y, por consiguiente, al borrar la línea es necesario volver a dibujarlo.

Aun cuando el final de cada línea dibujada desde la puerta para palanca de mando se detiene justo antes de la línea del horizonte, éste se ve afectado. Debido a la forma en que el Spectrum controla el color, la porción del horizonte más próxima al punto donde termina el rayo adquiere el mismo color que el utilizado para dibujar la línea. Ello se debe a que el Spectrum sólo puede soportar un color INK y un color PAPER dentro de una celda de carácter; cualquier gráfico que estuviera ya presente dentro de la celda tomará al color de primer plano del nuevo color INK empleado en la celda. Por consiguiente, además de volver a dibujar la puerta para palanca de mando, también se debe volver a dibujar la línea del horizonte después de borrar un rayo. La rutina continúa disparando rayos láser hasta que se efectúa una pulsación de tecla, en cuyo momento el control de programa retorna a la rutina de la puerta para palanca de mando principal, después de haber restablecido los colores INK y PAPER normales. Para llamar a esta subrutina se debe insertar esta línea:

```
3845 GOSUB 8000:REM IMAGEN PUERTA PALANCA MANDO
```

Pantalla ALU

```
7000 REM *** s/r imagen alu ****
7010 INK 6:PAPER 0:CLS
7015 :
7017 REM **** letra A ****
7020 FOR x=0 TO 55 STEP 5
7030 INVERSE 0
7040 FOR i=1 TO 2
7050 PLOT x,100
7060 DRAW 0,30
7070 DRAW 15,20
7080 DRAW 15,-20
7090 DRAW 0,-30
7095 DRAW 0,20
7096 DRAW -30,0
7110 IF x>55 THEN INVERSE 1
7115 NEXT i
7120 NEXT x
7130 :
7140 REM **** letra L ****
7150 FOR y=100 TO 150 STEP 5
7152 INVERSE 0
7155 FOR i=1 TO 2
7160 PLOT 113,y
7170 DRAW 0,-50
7180 DRAW 30,0
7190 IF y<150 THEN INVERSE 1
7200 NEXT i
7210 NEXT y
7220 :
7230 REM **** letra U ****
7240 FOR x=225 TO 170 STEP -5
7250 INVERSE 0
7260 FOR i=1 TO 2
7270 PLOT x,150
7280 DRAW 0,-50
7290 DRAW 30,0
7300 DRAW 0,50
7310 IF x>170 INVERSE 1
7320 NEXT i
7330 NEXT x
7340 :
7350 REM **** botones ****
7360 PRINT AT 10,7;"AND"
7370 PRINT AT 10,15;"OR"
7380 PRINT AT 10,22;"NOT"
7390 INK 3: CIRCLE 70,80,5
7400 INK 4:CIRCLE 128,80,5
7410 INK 5: CIRCLE 185,80,5
7420 :
7430 REM **** signo ? ****
7435 INK 6
7440 PLOT 113,45
7450 DRAW 0,15
7460 DRAW 30,0
7470 DRAW 0,-20
7480 DRAW -15,0
7490 DRAW 0,-7
7500 FOR r=6 TO 0 STEP -2
7510 CIRCLE 128,23,r
7520 NEXT r
7530 :
7540 IF INKEY$="" THEN GO TO 7540
7550 INK 0: PAPER 7: CLS
7560 RETURN
```

Pant. puerta para pal. mando

```
8000 REM **** s/r imagen puerta pal. mando ****
8010 INK 6: PAPER 0: CLS
8020 REM **** primer plano ****
8030 FOR n=1 TO 31
8040 PLOT 112+n,50
8050 DRAW 7*n-112,-50
8060 NEXT n
8070 :
8080 REM **** horizonte ****
8085 INK 6: INVERSE 0
8090 PLOT 0,50
8100 DRAW 255,0
8110 :
8120 REM **** puerta ****
8130 PRINT AT 1,18;"JOYSTICK PORT"
8140 PRINT AT 3,20;". . . . ."
8150 PRINT AT 5,21;". . . ."
8160 PLO 158,152
8170 DRAW 75,0
8180 DRAW 1,-1
8190 DRAW 1,-1
8200 DRAW 0,-1
8210 DRAW -1,-1
8220 DRAW -10,-25
8230 DRAW -2,-2
8240 DRAW -52,0
8250 DRAW -2,2
8260 DRAW -10,25
8270 DRAW -1,1
8280 DRAW -1,1
8290 DRAW 0,1
8300 DRAW 1,1
8310 :
8320 REM **** disparo ****
8340 INK RND*7
8350 LET x=RND*255-194
8360 LET y=-86
8365 INVERSE 0
8367 FOR i=1 TO 2
8370 PLOT 194,136
8380 DRAW x,y
8385 INVERSE 1
8387 NEXT i
8390 :
8400 REM **** comprobar tecla ****
8410 IF INKEY$="" THEN GO TO 8080
8415 INVERSE 0
8420 INK 0: PAPER 7: CLS
8430 RETURN
```

Patrones de enrejado

Veamos cómo se pueden crear las cuadrículas en que se basan los patrones en dos dimensiones

Si, en vez de trasladar el motivo que definimos en el capítulo anterior a lo largo de una línea, permitimos que se produzcan simultáneamente dos traslaciones no paralelas, entonces nuestro patrón se convierte en bidimensional. Comenzaremos a investigar este conjunto de patrones utilizando como unidad un único punto.

```
TO PUNTO
  PD FD 1 BK 1 PU
END
```

El procedimiento para llevar a cabo estas traslaciones simultáneas se define del siguiente modo:

```
TO CUADRICULA :PARTIDAX :PARTIDAY :XPASO
  :YPASO :ANGULO DRAW HT PU
  SETXY :PARTIDAX :PARTIDAY SETH 0
  REPEAT 3 [LINEA :XPASO ABAJO :YPASO
  :ANGULO]
END
```

Este procedimiento dibuja una cuadrícula de nueve puntos. Las entradas dan las coordenadas del punto de partida, el tamaño de los pasos X y Y, y la orientación de los puntos correspondientes de la siguiente fila desde la fila actual.

Éste es el procedimiento LINEA:

```
TO LINEA :X
  REPEAT 3 [UNIDAD SETX XCOR+:X]
  SETX XCOR-3*:X
END
```

dibuja una única línea de tres unidades a través de la pantalla, y después devuelve la tortuga a su punto de partida.

De momento, el procedimiento UNIDAD es simplemente un punto:

```
TO UNIDAD
  PUNTO
END
```

Otro procedimiento:

```
TO ABAJO :Y :A
  SETH :A
  FD :Y
  SETH 0
END
```

desplaza la tortuga hasta la fila siguiente (tal como lo hemos utilizado, ha supuesto el desplazamiento hacia "abajo" hasta la siguiente fila, de ahí el nombre del procedimiento), y luego restaura la orientación. En el diagrama se ofrecen los cinco tipos de enrejado plano, junto con los procedimientos en LOGO para dibujarlos.

A partir de combinaciones de estas cuadrículas básicas se pueden obtener muchos patrones diferentes, si bien probablemente sea más interesante modificar el procedimiento para que dibuje una línea en diversos ángulos en vez de recta a través de la pantalla.

Otra línea de investigación es ver cómo se puede perfeccionar la simetría de cada una de las cuadrículas añadiendo a la unidad dibujada en cada punto diversas formas de simetría. Existen 17 de tales patrones y todos ellos se pueden apreciar en el segundo diagrama. Un método obvio para dibujar estas 17 posibilidades consiste en reemplazar la instrucción UNIDAD del procedimiento LINEA por un procedimiento que dibuje la forma en ese punto.

La forma unidad

Las formas unidad se consiguen a partir de un motivo básico junto con diversas reflexiones y rotaciones. A modo de demostración, vamos a definir nuestro motivo básico, al cual llamaremos LIT. (Como antes, éste es de estado transparente y no emplea ningún procedimiento.)

```
TO LIT
  PD
  FD 15
  RT 90
  FD 5
  BK 5
  LT 90
  BK 15
  PU
END
```

Podemos utilizar los procedimientos que desarrollamos en el capítulo anterior para crear dos procedimientos: MOTIVO y su imagen de espejo, I.MOTIVO:

```
DEFINE "MOTIVO TEXTO "LIT
DEFINE "I.MOTIVO REESCRIBIR "LIT
```

Ahora podemos definir las formas unidad. Por ejemplo, las unidades para los patrones 7 y 17 serían las siguientes:

```
TO UNIDAD7
  LT 90 MOTIVO RT 90
END

TO UNIDAD17
  RT 30
  REPEAT 6 [MOTIVO I.MOTIVO RT 60]
  LT 30
END
```

Si vuelve a observar el procedimiento LINEA, verá que el mismo ejecuta el procedimiento UNIDAD. Por consiguiente, con el fin de dibujar el patrón 7, por ejemplo, se debe modificar el procedimiento UNIDAD de modo que rece UNIDAD7. Esto lo hacemos mediante el empleo de DEFINIR.UNIDAD7, donde:

```
TO DEFINIR.UNIDAD :NUM
  DEFINE "UNIDAD TEXT PALABRA "UNIDAD :NUM
END
```

Ahora ejecutaremos al mismo tiempo las partes de un patrón para dibujar la cuadrícula y dibujar la unidad.

Un procedimiento denominado PAT nos permite hacerlo:

```
TO PAT :CUADRICULA :NUM :PROC
   DEFINE "MOTIVO TEXT :PROC
   DEFINE "I.MOTIVO REESCRIBIR :PROC
   DEFINIR.UNIDAD :NUM
   RUN (LIST :CUADRICULA)
   ERASE MOTIVO
   ERASE I. MOTIVO
   ERASE UNIDAD
END
```

Para dibujar el patrón 17 ahora digitaremos:

```
PAT "HEX 17 "LIT
```

Esto dibuja una cuadrícula hexagonal, con UNIDAD17 en cada punto, utilizando LIT como motivo básico.

Este método funciona bien para todos los patrones, excepto para el 4, el 6, el 7 y el 12. En estos casos, la forma unidad no es la misma en cada punto, sino que sufre una transformación (reflexión, rotación o ambas a la vez). Una forma de tratar esta cuestión consiste en incorporar estas transformaciones a los procedimientos LINEA y ABAJO. De modo que vamos a definir TRANSX como la transformación a aplicar a la traslación básica a través de la pantalla, y TRANSY será la transformación a aplicar entre filas. Entonces LINEA y ABAJO se convierten en:

```
TO LINEA :X
   REPEAT 3 [UNIDAD SETX XCOR XCOR+:X
   TRANSX]
   SETX XCOR-3*:X
END

TO ABAJO :Y :A
   SETH :A
   FD :Y
   SETH 0
   TRANSY
END
```

Ahora definimos el patrón 7 como:

```
TO PATRON7 :PROC
   DEFINE "TRANSX[[][REFLEXION RT 180]]
   DEFINE "TRANSY[[][]]
   PAT "RECT 7 :PROC
   ERASE TRANSX
   ERASE TRANSY
END
```

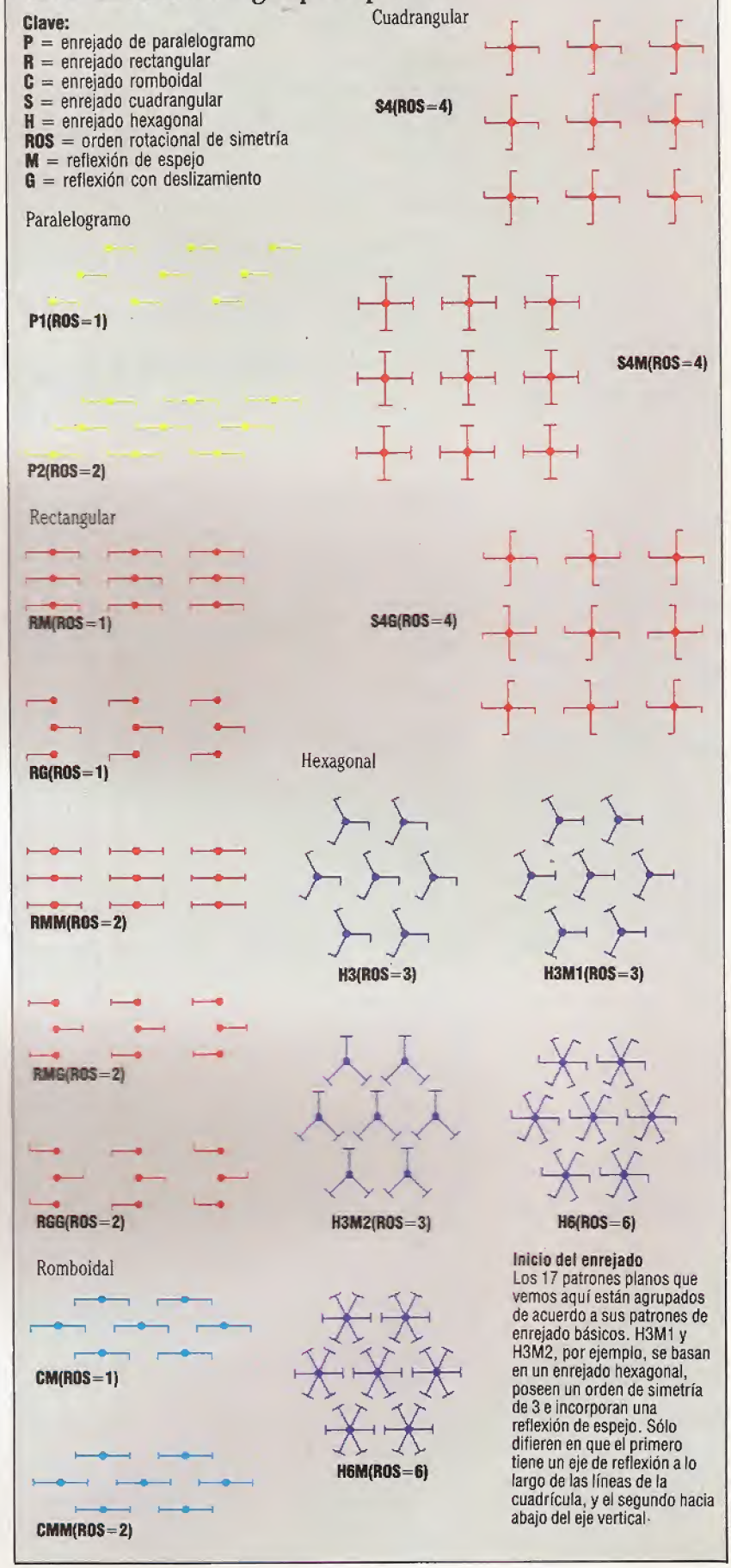
Para utilizarlo, entre PATRON7 "PATA. Tras ejecutar el procedimiento anterior, TRANSX se habrá definido como:

```
TO TRANSX
   REFLEXION
   RT 180
END
```

REFLEXION se utiliza para reflejar el patrón unidad. Este procedimiento se define reescribiendo el procedimiento UNIDAD:

Los diecisiete grupos planos

Clave:
P = enrejado de paralelogramo
R = enrejado rectangular
C = enrejado romboidal
S = enrejado cuadrangular
H = enrejado hexagonal
ROS = orden rotacional de simetría
M = reflexión de espejo
G = reflexión con deslizamiento



Inicio del enrejado
Los 17 patrones planos que vemos aquí están agrupados de acuerdo a sus patrones de enrejado básicos. H3M1 y H3M2, por ejemplo, se basan en un enrejado hexagonal, poseen un orden de simetría de 3 e incorporan una reflexión de espejo. Sólo difieren en que el primero tiene un eje de reflexión a lo largo de las líneas de la cuadrícula, y el segundo hacia abajo del eje vertical.

Ian McKinnell

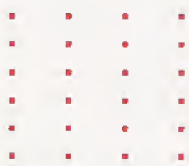
Los cinco tipos de enrejado de plano

Paralelogramo

```
TO PARALEL
  CUADR (-60) 90 80 50 205
END
```

Romboidal

```
TO ROMB
  CUADR (-30) 90 80 80 225
END
```



Rectangular

```
TO RECT
  CUADR (-80) 90 80 50 180
END
```

Cuadrangular

```
TO CUADRADO
  CUADR (-80) 90 80 80 180
END
```

Hexagonal

```
TO HEX
  CUADR (-30) 90 80 80 210
END
```

```
TO REFLEXION
  DEFINE "UNIDAD REESCRIBIR "UNIDAD
END
```

La reescritura supone ahora sustituir RT por LT, y viceversa, así como MOTIVO por I.MOTIVO y viceversa.

Nuestra versión previa del procedimiento de reescritura sólo intercambiaba RT y LT. Para modificar esto, todo cuanto hemos de hacer es modificar CAMBIAR.PALABRA, que ahora se convierte en:

```
TO CAMBIAR.PALABRA :PALABRA
  IF (ANYOF :PALABRA="RT :PALABRA="RIGHT)
  THEN OUTPUT "LEFT
  IF(ANYOF :PALABRA="LT :PALABRA="LEFT)
  THEN OUTPUT "RIGHT
  IF :PALABRA="MOTIVO THEN OUTPUT
  "I.MOTIVO
  IF :PALABRA="I.MOTIVO THEN OUTPUT
  "MOTIVO OUTPUT :PALABRA
END
```

Otra forma de abordar este problema sería utilizar la versión de REESCRIBIR que también modifica los subprocedimientos del procedimiento de entrada. Esta versión la ofrecemos entre las respuestas.

Para la mayoría de los patrones el movimiento entre puntos es simplemente una traslación, y no hay que llevar a cabo ninguna otra transformación. TRANSX y TRANSY, en consecuencia, no hacen nada. PATRON17 constituye un ejemplo de este tipo:

```
TO PATRON17 :PROC
  DEFINE "TRANSX[[]]
  DEFINE "TRANSY[[]]
  PAT "HEX 17 :PROC
  ERASE TRANSX
  ERASE TRANSY
END
```

Habiendo cubierto todas las posibilidades básicas, dejamos en sus manos la definición del resto de los patrones.

Complementos al LOGO

Para todas las versiones LCSI:

Emplee CS en lugar de DRAW
Emplee OR en lugar de ANYOF
SETPOS, seguido de una lista, se utiliza en lugar de SETXY
IF posee una sintaxis diferente:

```
IF :PALABRA=MOTIVO[OUTPUT "I.MOTIVO]
```

En el LOGO Atari TEXT y DEFINE no existen como primitivas, si bien en el manual se ofrece un método para definirlas

Respuestas a los ejercicios

1. Para hacer girar una forma alrededor del punto (X,Y) en un ángulo de A grados:

```
TO ROTAR :X :Y :A
  PU
  MAKE "O ORIENTACION
  MAKE "XANT XCOR
  MAKE "YANT YCOR
  MAKE "R SQRT (:XANT-:X)*(:XANT-
  :X)+(:YANT-:Y)*(:YANT-:Y)
  PU
  SETXY :X :Y
  SETH TOWARDS :XANT :YANT
  RT :A
  FD :R
  SETH :0+:A
  PD
END
```

2. Un procedimiento para reescribir que también reescriba subprocedimientos.

```
MAKE "PROCEDS.HECHOS[]

TO REESCRIBIR :PROC
  MAKE "PROCEDS.HECHOS FPUT :PROC
  :PROCEDS.HECHOS
  OUTPUT REESCRIBIR.PROC TEXT :PROC
END

TO REESCRIBIR.PROC :TEXTO
  IF :TEXTO=[]THEN OUTPUT[]
  OUTPUT FPUT REESCRIBIR.LINEA FIRST
  :TEXTO REESCRIBIR.PROC BUTFIRST
  :TEXTO
END

TO REESCRIBIR.LINEA :LINEA
  IF :LINEA=[] THEN OUTPUT[]
  IF LIST? FIRST :LINEA THEN OUTPUT FPUT
  REESCRIBIR.LINEA FIRST :LINEA
  REESCRIBIR. LINEA BUTFIRST :LINEA
  OUTPUT FPUT CAMBIAR.PALABRA FIRST
  :LINEA REESCRIBIR.LINEA BUTFIRST :LINEA
END

TO CAMBIAR.PALABRA :PALABRA
  IF (ANYOF :PALABRA="RT :PALABRA=
  "RIGHT)THEN OUTPUT "LEFT
  IF (ANYOF :PALABRA="LT :PALABRA=
  "LEFT) THEN OUTPUT "RIGHT
  IF PROCEDIMIENTO? :PALABRA THEN
  SUBPROCEDIMIENTO :PALABRA OUTPUT
  WORD "£:PALABRA
  OUTPUT :PALABRA
END

TO PROCEDIMIENTO?:NOMBRE
  IF NUMBER? :NOMBRE OUTPUT "FALSE
  IF LIST? :NOMBRE OUTPUT "FALSE
  TEST WORD? :NOMBRE
  IFTRUE IF WORD? TEXT :NOMBRE OUTPUT
  "FALSE ELSE IF NOT
  (TEXT :NOMBRE=[])
  OUTPUT "TRUE
  OUTPUT "FALSE
END

TO SUBPROCEDIMIENTO :PALABRA
  IF MEMBER? :PALABRA :PRODECS.HECHOS
  THEN STOP
  DEFINE (WORD "£:PALABRA)REEESCRIBIR
  :PALABRA
END
```

Adagio ma non troppo

Vamos a crear una rutina que desplace horizontalmente un dibujo de fondo en una pantalla del Commodore 64. Es obvia su utilidad en programas de juegos

El VIC o controlador de video del Commodore puede desplazar hasta ocho pixels la visualización en pantalla, en las dos direcciones. El desplazamiento horizontal se regula por medio de los tres bits inferiores correspondientes al registro del VIC situado en la posición 53270 ($D016). Si se van asignando valores progresivamente a estos tres bits del 7 al 0, la pantalla se va desplazando un pixel a la izquierda a cada valor. En BASIC se emplearía esta sentencia:

```
POKE 53270,(PEEK(53270)AND 248)+P
```

donde P tiene un valor entre 0 y 7.

Combinando esta facilidad con una rutina en código máquina para desplazar todos los datos de la pantalla una posición a la izquierda e introducir una nueva columna de datos en el margen derecho, es posible obtener un suave efecto de desplazamiento. La pantalla ha de reducirse a 38 columnas (en lugar de las normales 40 columnas) para que los datos aparezcan y desaparezcan de nuestra vista en un pausado desfile. El cambio al modo 38 columnas se realiza poniendo a cero el bit 3 del registro de los desplazamientos horizontales. En BASIC se hace así:

```
POKE 53270,PEEK(53270)AND247
```

La pantalla volverá a sus 40 columnas normales poniendo de nuevo el bit 3 a uno.

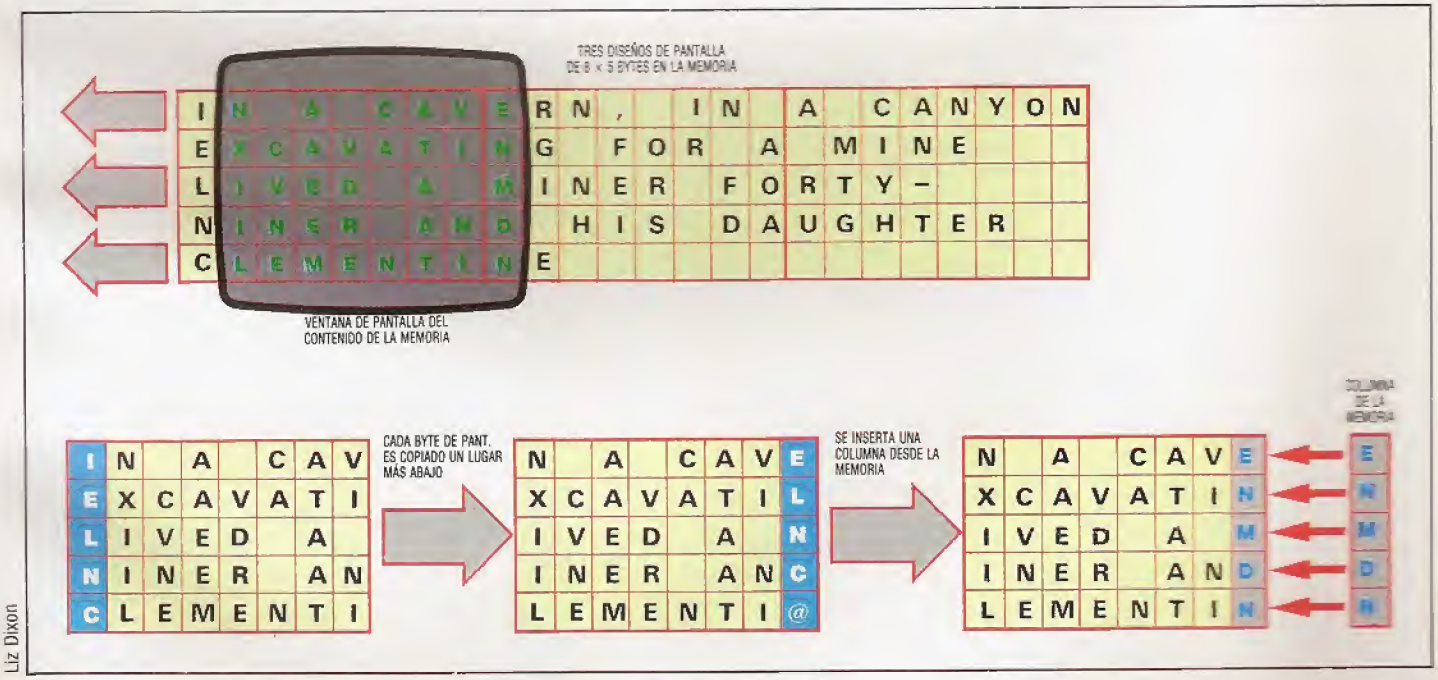
El diagrama de flujo muestra las diferentes tareas a realizar para producir ese suave movimiento horizontal. Importa añadir que si movemos o insertamos datos de pantalla, hay que hacer similares cambios en los datos del color.

Cambio de los datos de pantalla

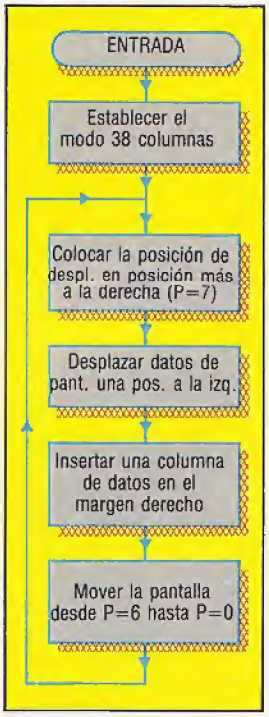
Se trata de una tarea en principio fácil. Los datos de pantalla se suelen guardar a partir de la posición 1024 ($0400): los primeros 40 bytes constituyen la fila superior, los siguientes 40 bytes la fila inmediatamente inferior, y así sucesivamente. Para dar la sensación de que los datos se mueven un lugar a la izquierda, basta con llevar cada byte de dichos datos al byte que se encuentra debajo de la posición original. Este fragmento de la rutina utiliza punteros de página cero y el direccionamiento indirecto para colocar cada byte de pantalla y color un byte más abajo en la memoria.

Llamando SB a la dirección *B*ase del área de la pantalla (*S*creen), la última posición de la fila superior será SB+39, la última de la fila inmediata inferior será SB+79, etc. Para que los datos a desplazar sobre la pantalla puedan almacenarse de modo similar en la memoria (o sea, en paquetes de 1 000 bytes), los datos a insertar por el lado derecho de la pantalla serán los bytes primero, 41-ésimo, 81-ésimo, etc., del área que reservamos para tales datos. Esto lo ilustramos con el siguiente dibujo:

Muévase a voluntad
El desplazamiento de las visualizaciones sobre la pantalla y desde la memoria pasa por tres fases principales. Primero, cada byte de la memoria de pantalla se mueve hacia abajo una posición. Debido al diseño de la pantalla, sus filas son tratadas en la memoria como bytes consecutivos. Esto produce el efecto en pantalla como si cada carácter se moviera un lugar a la izquierda, a excepción de los caracteres que aparecen en la columna extrema izquierda de la pantalla. En esta columna cada carácter parece como si apareciera en la misma posición pero en la columna extrema derecha. El carácter superior de la columna extrema izquierda desaparece durante el proceso dando entrada a un carácter extraño por la esquina inferior izquierda. La segunda fase se encarga de copiar la columna que interesa tomándola de la memoria y llevándola a la columna extrema derecha de la pantalla. Los registros del chip VIC a cargo del desplazamiento pueden accionarse para dar la sensación de que la pantalla se desplaza un pixel cada vez dentro del área visible



Liz Dixon



Liz Dixon

Poco a poco, suavemente
Para conseguir un movimiento suave podemos hacer uso de una facilidad particular del VIC. Este chip está dotado de unos registros especiales de desplazamiento que permiten a la pantalla visible moverse de su posición inicial respecto al encuadre. Es posible producir pixels individuales tanto en dirección horizontal como vertical. Combinando este efecto con el copiado de caracteres en código máquina, podemos obtener un suave movimiento sobre una pantalla reducida de 38 columnas

Al comienzo se establecerá un puntero que apunte al byte que está al comienzo del área de memoria que deseamos desplazar en pantalla. Una vez obtenido el desplazamiento, el puntero se incrementará en una unidad para que copie la columna siguiente en el margen derecho de la pantalla, desde donde puede desplazarse a la izquierda. Tras reiterar el proceso 40 veces consecutivas, se habrá obtenido un desplazamiento de todos los datos de la pantalla. El puntero será entonces incrementado en 960 unidades (1000-40) para apuntar al inicio de la pantalla siguiente.

Este proceso deberá ser repetido para el área de los datos de color. En resumen, haremos que la dirección de cada byte en el mapa del color tenga un desplazamiento hacia la dirección del byte correspondiente en el mapa de datos de pantalla. El proceso será repetido para tantas pantallas de datos como se hayan diseñado y guardado consecutivamente en memoria. Para utilizar la rutina del desplazamiento, hay que enviar algunas informaciones previas. La rutina necesita saber:

1) La dir. de inicio del área de memoria donde se guardan los datos de la pant. a desplazar.
2) El desplazamiento a los correspondientes datos de color.
3) El número de pantallas de datos a desplazar.
4) Un valor de retardo que sirva para ralentizar la operación del movimiento.

Estos datos han de ser colocados (POKE) en posiciones reservadas dentro del prog. en cód. máquina.

Prog. de llamada en BASIC

```
 10 REM ************************************
 20 REM ************************************
 30 REM **                                **
 40 REM **   PROGRAMA BASIC DE LLAMADA    **
 50 REM **     A RUTINA DESPLAZAMIENTO    **
 60 REM **                                **
 70 REM ************************************
 80 REM ************************************
 90 :
 95 DN=8 : REM PARA CASS DN=1
100 IF A=0 THEN A=1 : LOAD "SCROLL.HEX",DN,1
110 POKE 55,0 : POKE 56,32 : CLR : REM EXTREMO INFERIOR MEMORIA
115 REM GOSUB 1000 : REM ESTABLECE VISUALIZACION SIMPLE
120 :
130 LMEM    =49664:     REM INICIO MEMORIA
140 HMEM    =49665:     REM AREA
150 LCOFF   =49666:     REM DESPLAZ. AL COLOR
160 HCOFF   =49667:     REM MAPA
170 NMSCR   =49668:     REM NUMERO PANTALLAS (SCREENS)
190 DELAY   =49669:     REM VALOR DEL RETARDO (DELAY)
200 SCROLL  =49670:     REM DIRECCION INICIO PROGRAMA
210 :
220 REM PRINT CHR$(147) : REM BORRA PANTALLA
230 INPUT "DIRECCION INICIO EN DECIMAL";SA
240 HS=INT(SA/256):LS=SA-HS*256
250 POKE LMEM,LS: POKE HMEM,HS
260 :
270 INPUT "NUMERO PANTALLAS";NS
290 POKE NMSCR,NS
300 :
310 INPUT "DESPLAZ EN DECIMAL AL MAPA COLOR" ;OS
320 HO=INT(OS/256):LO=OS-HO*256
330 POKE LCOFF,LO:POKE HCOFF,HO
340 :
350 INPUT "VALOR DEL RETARDO>256";DV
360 IF DV>255 OR DV<0 THEN 350
370 POKE DELAY,DV
380 :
390 SYS SCROLL
400 POKE 53270,PEEK (53270) OR 8
```

El programa carga el código máquina en la memoria y pide la información necesaria a través de instrucciones INPUT. El programa secciona la información en la forma LO-HI donde sea necesario y la coloca (POKE) en los espacios de almacenamiento dispuestos al comienzo del programa. Posteriormente se llama a la rutina en código máquina.

Toda dirección de inicio, desplazamiento y número de pantallas deben ser especificadas, aunque los resultados no serán muy espectaculares si no se ha puesto ningún dibujo en el área de memoria especificada. Puede comprobarse el programa cargando y ejecutando el corto programa en BASIC que establece dos pantallas sencillas de datos que comienzan en la posición 8192. El desplazamiento al área de datos de color es 3 000 bytes. Para mover esta área de datos en la pantalla, hay que proporcionar la siguiente información en respuesta a las preguntas del programa que llama:

1) Dirección de inicio, en decimal: 8192
2) Desplazamiento del color: 3000
3) Número de pantallas: 2
4) Retardo: 255

Cargador en BASIC

```
 10 REM ************************
 15 REM **   CARGADOR EN BASIC  **
 20 REM **  PARA RUTINA DESPLAZ **
 30 REM **     HORIZONTAL       **
 40 REM ************************
 50 :
 60 FOR I=49670 TO 49945
 70 READ A : POKE I,A
 80 CC=CC+A
 90 NEXT
 92 READ CS:IF CS<>CC THEN PRINT "ERROR SUMA CONTROL":STOP
100 DATA173,22,208,41,247,141,22,208
110 DATA174,4,194,160,40,138,72,152,72
120 DATA173,22,208,41,248,24,105,7,141
130 DATA22,208,169,0,133,251,169,4,133
140 DATA252,169,0,133,253,169,216,133
150 DATA254,162,3,160,1,177,251,136
160 DATA145,251,200,177,253,136,145
170 DATA253,200,200,208,241,230,252
180 DATA230,254,177,251,198,252,136
190 DATA145,251,200,177,253,196,254
200 DATA136,145,253,230,252,230,254
210 DATA202,208,213,160,1,177,251,136
220 DATA145,251,200,177,253,136,145
230 DATA253,200,200,192,232,144,239
240 DATA173,0,194,133,253,173,1,194
250 DATA133,254,169,39,133,251,169,4
260 DATA133,252,32,241,194,173,0,194
270 DATA24,109,2,194,133,253,173,1,194
280 DATA109,3,194,133,254,169,39,133
290 DATA251,169,216,133,252,32,241,194
300 DATA162,6,173,22,208,41,248,141,22
310 DATA208,138,24,109,22,208,141,22
320 DATA208,172,5,194,136,208,253,202
330 DATA16,231,173,0,194,24,105,1,141
340 DATA40,194,173,1,194,105,0,141,1
350 DATA194,104,168,104,170,136,240,3
360 DATA76,19,194,173,0,194,24,105,192
370 DATA141,0,194,173,1,194,105,3,141
380 DATA1,194,202,240,3,76,17,194,96
390 DATA162,25,160,0,177,253,145,251
400 DATA202,240,29,165,251,24,105,40
410 DATA133,251,165,252,105,0,133,252
420 DATA165,253,24,105,40,133,253,165
430 DATA254,105,0,133,254,76,245,194
440 DATA96
450 DATA40227:REM*SUMA CONTROL*
```

Rutina para establecer la visualización

```
1000 REM **** ESTABLECE VISUALIZACION ****
1010 CL=3000:REM DESPLAZ AL MAPA COLOR
1020 SS=8192:REM INICIO MAPA VISUALIZ.
1030 FORI=SS TO SS+479
1040 POKE I,1:REM CODIGO PANT. DE LA "A"
1050 POKE+CL,1:REM BLANCO
1060 POKEI+480,2:REM CODIGO PANT. DE LA "B"
1070 POKEI+CL+480,14:REM AZUL CLARO
1080 NEXT
1085 FORI=SS+960TOSS+999
1090 POKEI,3:REM CODIGO PANT. DE LA "C"
1100 POKEI+CL,3:REM AZUL MARINO
1110 NEXT
1999 :
2020 SS=9192:REM SIGUIENTE INICIO PANT.
2030 FORI=SS TO SS+479
2040 POKEI,3:REM CODIGO PANT. DE LA "C"
2050 POKEI+CL,5:REM VERDE
2060 POKEI+480,4:REM CODIGO PANT. DE LA "D"
2070 POKEI+CL+480,0:REM NEGRO
2080 NEXT
2085 FORI=SS+960TOSS+999
2090 POKEI,5:REM CODIGO PANT. DE LA "E"
2100 POKEI+CL,2:REM ROJO
2110 NEXT
```

Movimiento horizontal

```
;++++++++++++++++
;++++++++++++++++
;++                ++
;++  MOV. HORIZONTAL  ++
;++                ++
;++ PARA COMMODORE 64 ++
;++                ++
;++++++++++++++++
;++++++++++++++++

SCRPTR=$FB              ;PAGINA 0
COLPTR=$FD              ;PUNTEROS COPIA

MEMPTR=$FD              ;PUNT P CERO DE MEMORIA
SCRLRG=$D016            ;REGISTRO DESPLAZ HORIZ
SCRNLO=$00              ;BYTE LO DE INICIO PANT
SCRNHI=$04              ;BYTE HI DE INICIO PANT
COLRLO=$00              ;BYTE LO DE INICIO COLOR
COLRHI=$D8              ;BYTE HI DE INICIO COLOR
BLOCKS=$03              ;BLOQUES DE BYTES 3*256
EXTRA=$E8               ;BYTES EXTRA HASTA 1000
NMCOLS=$28              ;NUM DE COLUMNAS
NMROWS=$19              ;NUM DE FILAS

*=$C200

MEMLO   *=*+1           ;INICIO MEMORIA QUE
MEMHI   *=*+1           ;SE HA DE DESPLAZAR
COFFLO  *=*+1           ;DESPLAZ AL MAPA DE COLOR
COFFHI  *=*+1
NMSCRN  *=*+1           ;NUM PANTALLAS
DELAY   *=*+1           ;VALOR BUCLE DEL RETARDO

;++++ ESTABLECE MODO 38 COLUMNAS ++++

        LDA SCRLRG
        AND #$F7
        STA SCRLRG

;++++ ESTABLECE PRIMERA POSICION A DESPLAZ ++++

        LDX NMSCRN
NEXSCR
        LDY #NMCOLS
START
        TXA
        PHA             ;COLOCA LOS REG X,Y
        TYA             ;EN LA PILA (PUSH)
        PHA

        LDA SCRLRG
        AND #$F8
        CLC
        ADC #$07
        STA SCRLRG

;++++ COPIA PANT/COLOR UN LUGAR IZQ ++++

        LDA #SCRNLO
        STA SCRPTR      ;ESTABLECE PAGINA 0
        LDA #SCRNHI
        STA SCRPTR+1    ;PUNTEROS PARA LA
        LDA #COLRLO
        STA COLPTR      ;COPIA
        LDA #COLRHI
        STA COLPTR+1

        LDX #BLOCKS
AGAIN
        LDY #$01
NEXT
        LDA (SCRPTR),Y
        DEY
        STA (SCRPTR),Y
        INY
        LDA (COLPTR),Y
        DEY
        STA (COLPTR),Y
        INY
        INY
        BNE NEXT

;++ TRASCRIBE BORDES PAGINA ++
        INC SCRPTR+1    ;INCR BYTES HI DE
        INC COLPTR+1    ;PUNTEROS PAGINA 0
        LDA (SCRPTR),Y
        DEC SCRPTR+1
        DEY
        STA (SCRPTR),Y ;TRASCRIBE PAGINA
        INY
        LDA (COLPTR),Y
        DEC COLPTR+1
        DEY
        STA (COLPTR),Y
        INC SCRPTR+1    ;INCR PUNTEROS P 0
        INC COLPTR+1    ;UNA VEZ MAS
        DEX
        BNE AGAIN

;++ PRODUCE BYTES EXTRA ++
        LDY #$01
ANTHER
        LDA (SCRPTR),Y
        DEY
        STA (SCRPTR),Y
        INY
        LDA (COLPTR),Y
        DEY
        STA (COLPTR),Y
        INY
        CPY #EXTRA
        BCC ANTHER      ;SI Y<EXTRA REPETIR

;++++ INSERTA COLUMNA DER DE PANT ++++

        LDA MEMLO
        STA MEMPTR
        LDA MEMHI       ;ESTABLECE PAGINA 0
        STA MEMPTR+1    ;PUNTEROS A LA MEMORIA
        LDA #NMCOLS-1
        STA SCRPTR      ;ESTABLECE PAGINA 0
        LDA #SCRNHI     ;PUNTEROS A LA PANTALLA
        STA SCRPTR+1
        JSR COPY40      ;COPIA COLUMNA
;++++ INSERTA COLUMNA DER DE COLOR ++++
        LDA MEMLO
        CLC
        ADC COFFLO      ;SUMA DESPLAZ AL
        STA MEMPTR      ;MAPA COLOR
        LDA MEMHI       ;ESTABLECIENDO LOS
        ADC COFFHI      ;PUNTEROS PAGINA 0
        STA MEMPTR+1
        LDA #NMCOLS-1   ;ESTABLECE PUNTEROS
        STA SCRPTR      ;PAGINA 0 AL COLOR
        LDA #COLRHI     ;RAM
        STA SCRPTR+1
        JSR COPY40      ;REALIZA LA COPIA

;++++ POSICIONES DEL 6 AL 0 PARA DESPLAZ ++++

        LDX #$06
MORE1
        LDA SCRLRG
        AND #$F8
        STA SCRLRG
        TXA
        CLC
        ADC SCRLRG
        STA SCRLRG

        LDY DELAY       ;CUENTA ATRAS
MORE2
        DEY             ;VALOR RETARDO
        BNE MORE2
        DEX
        BPL MORE1

;++++ INCREMENTA PUNT. MEMORIA ++++

        LDA MEMLO
        CLC
        ADC #$01
        STA MEMLO
        LDA MEMHI
        ADC #$00
        STA MEMHI

;++++ COMPRUEBA FIN AREA MEMORIA ++++

        PLA
        TAY             ;RECUPERA LOS REG X,Y
        PLA
        TAX             ;DE LA PILA
        DEY
        BEQ NOJMP
        JMP START
NOJMP
        LDA MEMLO
        CLC
        ADC #$C0        ;SUMA 1000-40
        STA MEMLO       ;AL PUNTERO MEMORIA
        LDA MEMHI
        ADC #$03
        STA MEMHI
        DEX
        BEQ RETRN
        JMP NEXSCR
RETRN
        RTS

;++++ S/R DE COPIA A CADA 40 BYTES ++++

COPY40
        LDX #NMROWS
        LDY #$00
REPEAT
        LDA (MEMPTR),Y
        STA (SCRPTR),Y
        DEX
        BEQ FINISH      ;AGOTADAS TODAS LAS FILAS?

        LDA SCRPTR
        CLC
        ADC #NMCOLS     ;INCR EN 40 LOS PUNTS
        STA SCRPTR
        LDA SCRPTR+1
        ADC #$00
        STA SCRPTR+1

        LDA MEMPTR
        CLC
        ADC #NMCOLS
        STA MEPTR
        LDA MEMPTR+1
        ADC #$00
        STA MEMPTR+1
        JMP REPEAT
FINISH
        RTS
```

"Mindstorms" (Ideas geniales), de Seymour Papert, Harvester Press, 1980

Iniciamos esta recensión comentando dos reveladoras obras que examinan las interioridades de la informática: "Mindstorms" (Ideas geniales), de Seymour Papert, y "The soul of a new machine" (El alma de una nueva máquina), de Tracy Kidder

"Mindstorms"

"The gears of my childhood" (Los engranajes de mi infancia) se titula el prólogo de *Mindstorms*; en él Seymour Papert explica su libro, sus ideas y su persona. En su vehemente monólogo nos habla de la fascinación que, cuando niño, ejercieron en él los

"Un nuevo mundo de informática personal está a punto de aparecer... inseparable de la historia de quienes lo realizarán."

trenes de engranajes y como, a través de ellos, descubrió el placer de aprender; ahora continúa enamorado de la moderna encarnación de éstos: el ordenador personal en el cuarto de juegos.

El estilo de Papert y su propia personalidad son dominantes, pero el libro se halla imbuido de la personalidad y las ideas de Jean Piaget (1896-1980), el psicólogo-pedagogo que más influencia ha ejercido en los tiempos modernos. El LOGO, el lenguaje para ordenadores desarrollado y descrito a lo largo del libro, es en realidad el *hommage au maître* de Papert, un intento por formular una expresión con-

"Empleo la imagen de Mathland para desarrollar mi idea de que la presencia del ordenador podría conllevar la unión de las culturas humanista y matemático-científica."

creta de las ideas de Piaget acerca del niño como "constructor activo de sus propias estructuras intelectuales".

En este sentido, el libro no versa en realidad ni sobre LOGO ni sobre Piaget, ni siquiera sobre el propio Papert; su verdadero tema es cómo los ordenadores pueden crear espacios de aprendizaje ("micromundos") en los cuales el niño pueda aprender a pensar y razonar con la misma alegría y la misma riqueza con que lo hizo Papert con sus engranajes alegóricos.

Al igual que en el caso de Piaget y su obra escrita, Papert y el LOGO han sido alabados por sus difusores y luego criticados por los revisionistas en el espacio de apenas unos pocos años. Las ideas de Piaget acerca de los distintos estadios del desarrollo del niño han sido cuestionadas por su aparente determinismo; mientras que en la actualidad hay quienes piensan que el LOGO es útil sólo para enseñar geometría y programación, en vez de considerarlo como la "piedra filosofal".

El título, *Mindstorms* (Ideas geniales), describe y define el libro. Sin duda alguna, Seymour Papert escribe tal como piensa: en una mezcla maravillosamente estimulante de frío análisis académico y ardoroso tono profesoral.

"The soul of a new machine" (El alma de una nueva máquina), de Tracy Kidder, Penguin, 1982

"The soul of a new machine"

Tracy Kidder nos cuenta la historia secreta del desarrollo del miniordenador Eagle, de Data General, empresa llevada a cabo a partir de cero en el transcurso de apenas un año. El libro obtuvo el premio Pulitzer de 1982 para obras ajenas al género de ficción y ha servido de base a una película. Posee misteriosas insinuaciones psicológicas para los seguidores de Piaget, desde los rostros con cierto aire de robot de los jóvenes científicos de la cubierta del libro, pasando por los ecos edípicos de los esfuerzos de la joven empresa de ordenadores por superar a su gigantesca casa madre, DEC, hasta la figura de caracteres novelescos de West, el ingeniero de proyectos y jefe del equipo. Se la podría definir como la versión informática de *Moby Dick*, la novela de Melville, una historia que cauti-

"Monótonamente, el ordenador... estaba contando una antigua historia: el materialista cuento de hadas hecho realidad."

va al lector, narrada con sencillez.

El libro es técnicamente espléndido, con descripciones gráficas de cada una de las etapas del diseño y la construcción del miniordenador de 32 bits. Kidder pudo observar directamente gran parte del desarrollo del proyecto y relata, en una prosa austera y cristalina, las explicaciones para las decisiones adoptadas y las acciones emprendidas.

El libro versa fundamentalmente sobre la existencia brillante y algo aislada de los técnicos cuyo microcosmos está compuesto por los sistemas operativos que ellos mismos inventan. Kidder se siente igualmente fascinado por sus reacciones personales ante el desarrollo de la máquina, por la enmarañada atmósfera que envuelve al grupo, en que se mezclan lealtad, presiones, manipulación, y por la bizantina política de la empresa, que determina la

"Ahora el juego era diferente... La máquina ya no pertenecía a quienes la habían creado."

planificación y el diseño de la máquina. El pararrayos de toda esta energía es Tom West, a quien en el prólogo de la obra se describe literalmente como "un buen hombre en medio de una tormenta". Su equipo exclusivo de jóvenes ingenieros trabaja toda la noche, inspirado en parte por su ejemplo, a pesar de lo cual él cínicamente les niega una máquina de prueba esencial porque "no pagar las horas extras sale más barato que el nuevo equipo". West bien puede ser el capitán Achab o el capitán América... Tracy Kidder no parece tener muy claras las ideas en este sentido. Pero su libro arroja luz sobre algunos de estos hombres.

Guía del comprador

¿Cuáles son las preguntas que el usuario debería formular para adquirir el equipo más adecuado? Aquí se las sugerimos

Cuando usted eligió su ordenador, quizá fue aconsejado en el sentido de que decidiera primero el software que deseaba utilizar y optase luego por el hardware necesario para ejecutarlo. Hasta cierto punto, éste es un buen consejo cuando se trata de elegir un sistema para comunicaciones. Sin embargo, creemos que a menudo es muchísimo mejor adquirir el modem y el software como paquete global en el mismo comercio.

Lo primero que debe hacer es decidir para qué utilizará el sistema. Para acceder a sistemas de videotexto tales como el Prestel necesitará un modem de 1200/75 baudios; para la mayoría de los sistemas de tablones de anuncios y correo electrónico precisará un modem de 300 baudios, y para el trabajo directo usuario-usuario se recomienda una velocidad de 1200 baudios. Para acceder a Compunet se requiere un modem Compunet especial, puesto que el software está almacenado en la ROM del modem. Usted tendrá, asimismo, que tener en cuenta las frecuencias utilizadas. En Gran Bretaña y España, por ejemplo, necesitará frecuencias CCITT; en Estados Unidos se emplean tonos Bell. Por consiguiente, si desea realizar llamadas transoceánicas directas, es preciso que su modem posea ambas frecuencias.

Para las comunicaciones usuario-usuario es sumamente aconsejable tener un modem que pueda conmutar entre frecuencias de emisión y de recepción. Si éste opera sólo en la de emisión, el modem al cual llame deberá ser capaz de conmutarse a la de respuesta.

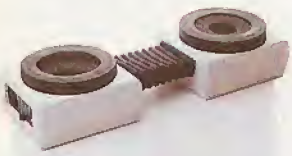
Si pretende llamar a tablones de anuncios, resulta esencial un modem con la facilidad de *llamada automática* (no disponible en España). Ello se debe a que la mayoría de los tablones de anuncios poseen únicamente una línea telefónica y sólo permiten el acceso a un usuario a la vez. Por este motivo con frecuencia están comunicando, de modo que es lógico dejar que el modem realice la llamada repetidamente por usted. Un modem con llamada automática también debe soportar software capaz de obtener un número (ya sea desde el teclado o bien desde una base de datos de números de teléfono) y enviarlo al modem en el formato correcto. Lamentablemente, los diferentes modems con llamada automática esperan que el número que se les envíe esté en formatos diferentes, de modo que es necesario que el modem y el software sean compatibles: un buen argumento para adquirir tanto el modem como el software en un solo paquete.

Steve Cross

Si desea que la gente le envíe datos directamente, hallará muy justificada la inversión que supone la adquisición de un modem con *contestador automático*. No obstante, si piensa utilizar para este fin su línea telefónica normal, deberá tener la cortesía de advertírselo a sus amigos, en especial a aquellos que no posean un modem. De lo contrario, ¡su modem podría silbarles durante diez segundos y después colgar!

El software adecuado también es esencial para un modem con contestador automático. Esta clase de software comprende desde paquetes capaces de abrir un nuevo archivo para cada llamada y guardarlo en disco, hasta software sofisticado para tablones de anuncios tales como el TBBS. Un interesante ejemplo de software con contestador automático para micros CP/M es el Remote CP/M. Este paquete permite que usted "disque" su micro CP/M y ejecute cualquier programa CP/M a través del te-

Unidad ideal
La gama disponible de opciones de hardware y software hacen que adquirir un modem se convierta en una tarea confusa y especializada: nuestra unidad ideal (reflejada en la ilustración) combina las características de los numerosos modems que existen en la actualidad. Para un principiante, el mejor enfoque es especificar por anticipado sus exigencias en cuanto a comunicaciones y confiar en el vendedor para que elija por él el paquete adecuado



Protek 1200
Tipo: Modem acústico
Velocidad: 1200/75 y 1200/1200 baudios

Prism 1000
Tipo: Modem compacto para videotexto
Velocidad: 1200/75 baudios

Commodore Communications Modem
Tipo: Modem compacto de videotexto para Compunet
Velocidad: 1200/75 y 1200/1200 baudios

Ian McKinnell

Epson CX-21
Tipo: Modem acústico
Velocidad: 300 baudios

léfono, lo que resulta ideal para aquellos usuarios que posean un micro de sobremesa o uno portátil.

Al elegir el software apropiado, es casi seguro que desee un paquete que sea capaz de recibir y emitir archivos ASCII. Asegúrese de que el paquete soporte cualquier dispositivo de almacenamiento que usted utilice. Los programas en BASIC se pueden transmitir en forma ASCII, como ya sabemos; pero si desea transmitir archivos binarios (p. ej., archivos .COM CP/M), necesitará alguna clase de protocolo para transmisión binaria. De éstos, el más difundido es el XModem.

También es conveniente poder crear archivos de *conexión automática* para distintos sistemas. Entonces, cuando conecte con un sistema, todo cuando deberá hacer es cargar el archivo apropiado, conteniendo su identificador, contraseña, etc. Algunos sistemas con llamada automática enlazarán este tipo de archivos con una base de datos de números de teléfono, de modo que lo único que ha de hacer es entrar el nombre del servicio que desea; el software buscará el número de teléfono, lo marcará y se conectará automáticamente.

Una vez decidido respecto a las características que requiere, ha de encontrar un paquete de modem y software que soporte estas facilidades. Puede adquirir el modem y el software por separado, pero le aconsejamos decididamente que le proporcione al vendedor una lista de las características que desea, así como detalles sobre el micro que utilizará, y deje que él le busque un paquete completo compuesto por modem, cable y software. De esta forma, si el sistema no cumple con lo que deseaba que hiciera, tanto usted como el comerciante sabrán quien debe arreglar la situación.

Elección de un terminal idóneo

Si ya posee un micro, es probable que desee usarlo como terminal. Ello será factible independientemente de la máquina que posea (si está muy decidido, incluso se puede utilizar un ZX81), si bien algunos micros se prestan mejor que otros para las comunicaciones. He aquí un breve resumen de la idoneidad de los cuatro micros más populares.

Con mucho, la máquina más fácil de convertir en terminal es el BBC Micro. De hecho, usted puede escribir un simple programa de terminal mudo para el mismo con unas pocas líneas de BASIC:

```
100 REM Programa de terminal mudo para el BBC
110 *FX2,2
120 *FX3,1
130 REPEAT: GET A$: IFA$=CHR$(13) THEN PRINT
140 PRINT A$;:UNTIL FALSE
```

Para el BBC Micro existen varios buenos paquetes de comunicaciones, algunos de los cuales se suministran en ROM, pero suelen ser caros. La mayoría ofrecen todas las características requeridas por el usuario medio, porque la mayor parte del trabajo lo realiza el sistema operativo del ordenador: todo lo que ha de hacer el programador es añadir los toques finales.

El Spectrum es más difícil de adaptar. En primer lugar, no podrá batir ningún récord de velocidad en materia de comunicaciones desde BASIC. Con un programa en este lenguaje apenas es posible emplear el Spectrum hasta unos 10 baudios y entonces no será capaz de llevar a cabo tareas tales como almacenar los caracteres en RAM. También puede olvidarse de su Interface 1: no es una interface RS232 y sirve de poco para comunicaciones. Para el Spectrum todavía no existe virtualmente software para comunicaciones.

El Commodore 64 también posee una interface en serie no estándar y la mayoría de los modems para la máquina se enchufan en la puerta para el usuario. Nuevamente, tampoco podrá hacer nada muy útil en BASIC. Asimismo, el 64 posee un juego de caracteres ASCII no estándar, de modo que el software para comunicaciones ha de traducir entre ASCII estándar y ASCII Commodore, algo que se puede hacer muy fácilmente empleando una tabla de referencia. No existen paquetes de software de comunicaciones aprobados por Commodore. Los usuarios británicos tienen la posibilidad de obtener el Termulator de Chris Townsend Computers; los norteamericanos habrán de consultarlo con las tiendas locales. Los usuarios de Compunet pueden utilizar el modem Compunet Commodore oficial con software residente, pero este modem es exclusivo para Compunet.

Tandy suministra software de terminal mudo para sus máquinas TRS-80 basadas en disco que operan bajo la mayoría de los sistemas operativos. La mayor parte de este software está destinado a la transferencia directa máquina-máquina, pero también se puede emplear con modems. Las máquinas Tandy fueron los primeros micros que se utilizaron para ejecutar y acceder a tablones de anuncios, de modo que normalmente se puede hallar una buena selección de software para comunicaciones de dominio público, basado tanto en disco como en cassette. Las TRS-80 no son aptas para operación de videotexto (1200/75 baudios).

En realidad cualquier micro de gestión se puede utilizar para comunicaciones, pero existen varios puntos importantes a tener en cuenta. En primer lugar, existe una creciente tendencia hacia los modems incorporados; éstos (en particular aquellos con software de comunicaciones basado en ROM) son los más fáciles de emplear. Tan sólo es necesario enchufarlos en el conector del teléfono.

Si se desestima la opción del modem incorporado, la siguiente mejor opción es un micro con al menos dos puertas RS232, que permitirá el uso simultáneo de un modem y una impresora en serie. Algunos micros poseen puertas para modem separadas y no estándares: éstas servirán siempre y cuando usted pueda conseguir un cable adecuado para el modem que elija.

Desde el punto de vista del software, no tendrá ningún problema con máquinas CP/M, MS-DOS o PC-DOS. Los sistemas operativos no estandarizados, sin embargo, corren tanto riesgo desde el punto de vista de los paquetes para comunicaciones como con cualquier otro tipo de software.

Si las comunicaciones son su principal razón para adquirir un micro, las llamadas máquinas "de regazo", tales como el Tandy Modelo 100, el NEC PC8201A y el Olivetti M10 son muy interesantes. Con una de ellas y un modem que funcione con pilas tendrá un terminal de maletín convenientemente portátil. Las tres máquinas poseen editores de texto incorporados y software de terminal y permiten unas 20 horas de uso con cuatro pilas AA.

Tomar medidas

Diseñaremos un fragmento de software para que el robot pueda ubicar y medir con exactitud el lado de un objeto

Conseguir que nuestro robot localice y mida el lado de un objeto exige un software algo complejo. El robot investigará el objeto empleando los sensores de microinterruptor que ya le hemos instalado. Las primeras ideas que tuvimos para un posible método de realizar esta tarea fueron:

1) Hallar el objeto.
2) Encontrar un extremo del lado localizado.
3) Ir probando a lo largo del lado del objeto hasta hallar el otro extremo.

La primera etapa se puede cubrir fácilmente si suponemos que cuando comienza el programa el robot está orientado hacia el lado del objeto que deseamos medir. El principal problema que se puede prever es que el robot alcance uno de los extremos del lado con un solo sensor en vez de hacer contacto con ambos. En el diagrama se reflejan las posibles variaciones. No obstante, aun cuando uno de los sensores delanteros estuviera cerrado, se puede saber si se trata del izquiedo o del derecho y, por consiguiente, podemos desarrollar una estrategia para tratar esta situación.

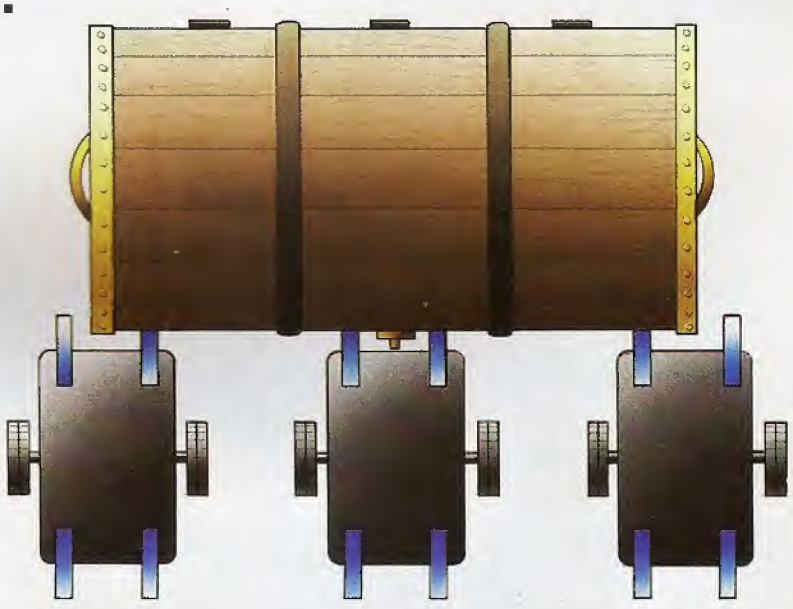
Debemos, asimismo, dar por sentado que inicialmente el robot está posicionado en 90° respecto al lado del objeto a medir. Con ello nos ahorramos el tratar casos en los cuales el robot efectúe un contacto oblicuo con el objeto.

La segunda etapa de nuestro método se simplifica decidiendo que el robot siempre avance hacia el extremo derecho del objeto antes de comenzar a medirlo. Para localizar el extremo derecho, el robot debe "sentir" su recorrido a lo largo del lado, desplazándose en pasos discretos hacia la derecha hasta que sólo se cierra el sensor izquierdo (y no ambos). Para ir sondeando el lado, el robot ha de llevar a cabo una serie complicada de maniobras, implicando cada paso cinco movimientos. Suponiendo que inicialmente el robot esté en contacto con el lado del objeto, debe primero retroceder, luego girar 90°, avanzar una cierta distancia, volver a girar 90° y por último avanzar hasta que los sensores vuelvan a hacer contacto con el objeto. El diagrama ilustra los pasos que supone la maniobra completa. La *longitud de paso* (distancia entre un punto de contacto con el lado del objeto y el siguiente punto) es equivalente a la parte de la maniobra en la cual el robot se desplaza paralelamente al lado del objeto.

Para localizar con precisión el extremo derecho del objeto, podría parecer necesario que el robot sondeara el lado en pasos de unos pocos milímetros, pero no es así. Por el contrario, podemos utilizar pasos mayores, sondeando el lado del objeto hasta sobrepasar el extremo, y luego retroceder un paso para sondear en pasos menores para localizar

Sensaciones...

Este diagrama ilustra las tres alternativas que se pueden producir cuando los sensores entran en contacto con el lado de un objeto. Cuando sólo está cerrado el sensor de la derecha, el robot ha detectado el extremo izquierdo del objeto; si ambos están cerrados, "sabe" que se halla en algún punto del medio; si sólo está cerrado el sensor de la izquierda, se ha topado con el extremo derecho del lado



Cerrado sólo el sensor derecho | Cerrados ambos sensores | Cerrado sólo el sensor izquierdo

Kevin Jones

con exactitud el extremo derecho. Una longitud de paso adecuada es la distancia entre los dos sensores delanteros (alrededor de 60 mm), puesto que ello asegura que cuando se sobrepase un extremo, uno de los sensores se cierre.

La tercera etapa supone un procedimiento de sondeo similar, pero en esta ocasión el robot se desplaza hacia la izquierda, contando la cantidad de pasos dados hasta llegar al extremo izquierdo. Al final de esta etapa, la longitud del lado del objeto estará almacenada en la variable del contador y podrá ser impresa.

Programa de medición

Ofrecemos listados para el Commodore 64 y el BBC Micro. Se deben insertar los coeficientes impulso/distancia e impulso/ángulo para su propio robot (hallados mediante la experimentación en el capítulo anterior del proyecto del robot). La estructura de procedimientos del BASIC BBC es ideal para escribir un programa de este tipo. Podemos adoptar un enfoque sumamente estructurado para este problema, controlando, mediante procedimientos separados, cada movimiento que efectúe el robot. Dos características del BASIC BBC (los nombres de variables ampliados y el paso de parámetros entre procedimientos) hacen que el programa se asemeje mucho a la forma en que nosotros pensamos. La versión Commodore puede adoptar el mismo enfoque estructurado, pero su estructuración en BASIC resulta mucho más dificultosa, lo que hace que el programa sea mucho más difícil de seguir que la versión para el BBC.

Habiendo aislado las principales tareas que debe llevar a cabo el programa, podemos diseñar procedimientos individuales para desplazar el robot y combinar una serie de maniobras para crear un método de *sondeo*. La combinación de los procedimientos de sondeo forma el procedimiento de *medición* global. En esta aplicación, los distintos niveles de procedimientos se pueden identificar fácilmente, yendo desde un sencillo método para impulsar los motores, en el nivel inferior, hasta la actividad de medición completa en el nivel más elevado.

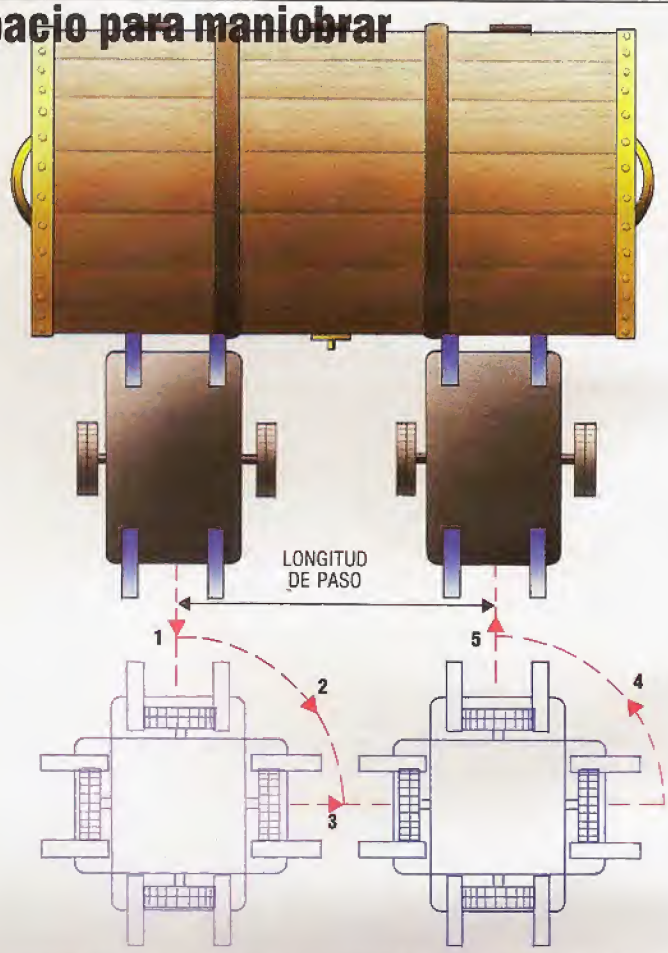
Si inicialmente no se coloca el robot exactamente en 90° respecto al lado del objeto a medir, pueden surgir problemas. Si sólo uno de los sensores hace contacto cuando ambos deberían hacerlo, entonces la lógica del programa hará que el robot decida que está posicionado en uno de los extremos del objeto. Si esto sucede, interrumpa el programa, alinee al robot perpendicularmente al lado a medir, y vuelva a ejecutar el programa desde el principio.

Se pueden identificar varios errores intrínsecos de medición. La anchura de cada sensor, por ejemplo, es de unos 5 mm. La localización de los extremos izquierdo y derecho del objeto puede, por consiguiente, producir un error máximo total de 10 mm.

Además, cuando el robot sondea con precisión los extremos del objeto, lo hace en pasos de 5 mm y, por lo tanto, se puede introducir aún otro error de 10 mm.

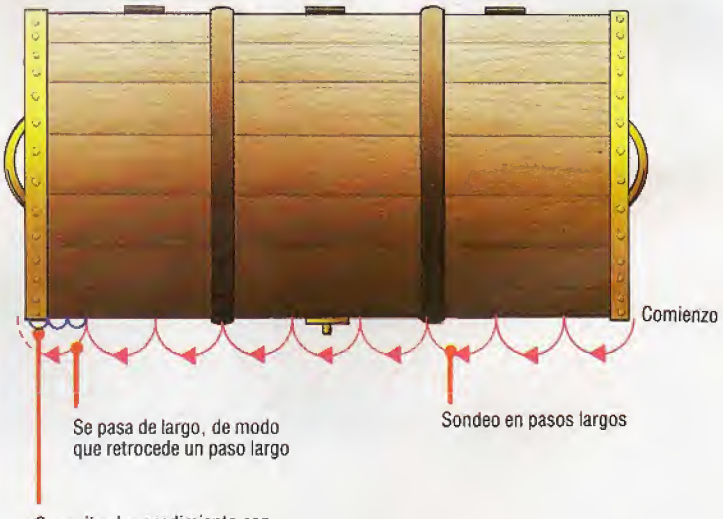
En las pruebas con nuestro robot prototipo, el promedio de error para la medición de un objeto de 410 mm de lado fue de unos 20 mm: un margen de sólo el 5 %.

Espacio para maniobrar



La figura superior muestra la maniobra de sondeo básica del robot. Cuando ambos sensores están cerrados, el robot retrocede (1) para poder girar sin chocar contra el objeto.
El robot efectúa luego un giro de 90° (2) y se desplaza hacia adelante cubriendo la longitud de paso (3). El robot realiza otro giro de 90°, de modo que queda nuevamente encarado al objeto (4) y, por último, avanza (5) para comprobar si el objeto se halla todavía enfrente de él.
Este proceso se repite en toda la longitud del objeto, como podemos ver abajo. Al principio los pasos del robot son largos, para obtener una longitud aproximada del objeto. Cuando sólo uno de los sensores, o ninguno, está cerrado, el robot desanda el último paso y repite en su totalidad el procedimiento de sondeo con pasos más cortos

Procedimiento de sondeo



Kevin Jones

Listado para BBC Micro

```
1000 REM **** MEDICION ROBOT BBC ****
1010 MODE 7
1020 PROCinicializar
1030 PROCmedir
1040 PROCimprimir
1050 END
1060 DEF PROCmedir
1070 PROChallar
1080 REM ** SOLO UN PARACHOQUES ? **
1090 PROCcomprobar_parachoques
1100 REM **** HALLAR EXTREMO ****
1110 REPEAT:PROCsondear(derecha,anchura)
1120 UNTIL (?REGDAT AND 192)=parachoques_izquierdo
1130 REM ** RETROCEDER Y AVANZAR HASTA EL EXTREMO **
1140 PROCsondear(izquierda, anchura)
1150 REPEAT:PROCsondear(derecha,anchura_pequeña)
1160 UNTIL (?REGDAT AND 192)=parachoques_izquierdo
1170 PRINT"HALLADO EXTREMO DERECHO"
1180 PRINT"COMIENZA LA MEDICION"
1190 REM ** EMPEZAR A MEDIR **
1200 contador=anchura
1210 REPEAT:PROCsondear(izquierda,anchura)
1220 contador=contador+anchura
1230 UNTIL (?REGDAT AND 192)=parachoques_derecho
1240 REM ** RETROCEDER Y AVANZAR HASTA EL EXTREMO **
1250 contador=contador-anchura
1260 PROCsondear(derecha,anchura)
1270 REPEAT:PROCsondear(izquierda,anchura_pequeña)
1280 contador=contador+anchura_pequeña
1290 UNTIL (?REGDAT AND 192)=parachoques_derecho
1300 ?REGDAT=0
1310 ENDPROC
1320 :
1330 DEF PROCimprimir
1340 CLS
1350 PRINTTAB(5,12)"EL LADO DEL OBJETO
     MIDE  ";contador;" mm"
1360 ENDPROC
1370 :
1380 DEF PROCinicializar
1390 RDD=&FE62:REGDAT=&FE60
1400 ?RDD=15:REM LINEAS 0-3 SALIDA
1410 ?REGDAT=1:REM ENCENDER BIT PUESTA A CERO
1420 adelante=4:atras=2:izquierda=6:derecha=0
1430 coeficiente_id=3.34446:coeficiente_ia=375/90
1440 parachoques_derecho=128:parachoques_izquierdo=64
1450 ambos_parachoques=0:ningun_parachoques=192
1460 anchura=60:anchura_pequeña=5
1470 ENDPROC
1480 :
1490 DEF PROCbuscar(sentido)
1500 REPEAT:PROCsondear(sentido,anchura)
1510 UNTIL (?REGDAT AND 192)=ambos_parachoques
1520 ENDPROC
1530 :
1540 DEF PROChallar
1550 REPEAT:PROCmover(adelante,8)
1560 UNTIL (?REGDAT AND 192)<>ningun_parachoques
1570 ENDPROC
1580 :
1590 DEF PROCcomprobar_parachoques
1600 IF (?REGDAT AND 192)=parachoques_derecho THEN
     PROCbuscar(derecha):ENDPROC
1610 IF (?REGDAT AND 192)=parachoques_izquierdo THEN
     PROCbuscar(izquierda):ENDPROC
1620 ENDPROC
1630 :
1640 DEF PROCsondear(direccion,paso)
1650 IF direccion=derecha THEN direccion_op=izquierda ELSE
     direccion_op=derecha
1660 PROCmover(atras,30)
1670 PROCgirar(direccion,90)
1680 PROCmover(adelante,paso)
1690 PROCgirar(direccion_op,90)
1700 REPEAT:PROCmover(adelante,8)
1710 UNTIL (?REGDAT AND 192)<>ningun_parachoques
1720 ENDPROC
1730 :
1740 DEF PROCmover(dir,distancia)
1750 ?REGDAT=(?REGDAT AND 1)OR dir
1760 impulsos=coeficiente_id*distancia
1770 FOR I=1 TO impulsos:PROCimpulso:NEXT I
1780 ENDPROC
1790 :
1800 DEF PROCgirar(dir,angulo)
1810 ?REGDAT=(?REGDAT AND 1)OR dir
1820 impulsos=coeficiente_ia*angulo
1830 FOR I=1 TO impulsos:PROCimpulso:NEXT I
1840 ENDPROC
1850 DEF PROCimpulso
1860 ?REGDAT(?REGDAT OR 8)
1870 ?REGDAT=(?REGDAT AND 247)
1880 ENDPROC
```

Listado para Commodore 64

```
  10 REM **** MEDICION ROBOT CBM ****
  20 GOSUB1000:REM INICIALIZAR
  30 GOSUB2000:REM MEDIR
  40 GOSUB3000:REM IMPRIMIR
  50 END
  60 :
1000 REM **** INICIALIZAR ****
1010 RDD=56579: REGDAT=56577
1020 POKE RDD,15:REM LINEAS 0-3 SALIDA
1030 POKE REGDAT,1:REM ENCENDER BIT PUESTA A CERO
1040 FW=4: BW=2: LF=6: RT=0
1050 PD=3.34446:PA=375/90
1060 RB=128:LB=64:BB040:NB=192
1070 WD=80:SW=5
1080 RETURN
1090 :
2000 REM **** MEDIR ****
2010 GOSUB3500:REM HALLAR OBJETO
2020 GOSUB4000:COMPROBAR PARACHOQUES
2030 REM ** HALLAR EXTREMO **
2040 WY=RT:SP=WD:GOSUB6000:REM SONDEAR
2050 IF(PEEK(REGDAT)AND192)<>LB THEN 2040
2060 REM ** RETROCEDER Y AVANZAR HACIA EXTREMO **
2070 DR=LF:DS=WD:GOSUB6000:REM SONDEAR
2080 DR=RT:DS=SW:GOSUB6000:REM SONDEAR
2090 IF(PEEK(REGDAT)AND 192)<>LB THEN 2080
2100 PRINT"HALLADO EXTREMO DERECHO"
2110 PRINT"COMIENZA LA MEDICION"
2120 REM ** EMPEZAR A MEDIR **
2130 CC=WD
2140 DR=LF:DS=WD:GOSUB6000:CC=CC+WD
2150 IF(PEEK(REGDAT)AND 192)<>RB THEN 2140
2160 REM ** RETROCEDER Y AVANZAR HACIA EXTREMO **
2170 CC=CC-WD
2180 DR=RT:DS=WD:GOSUB6000:CC=CC+SW
2190 IF(PEEK(REGDAT)AND 192)<>RB THEN 2180
2200 POKE REGDAT,0
2210 RETURN
2220 :
3000 REM **** IMPRESION ****
3010 PRINTCHR$(147)
3020 PRINT"EL LADO DEL OBJETO MIDE  ";CC; "MM"
3030 RETURN
3040 :
3500 REM **** HALLAR ****
3510 DR=FW:DS=5:GOSUB7000:REM MOVER
3520 IF(PEEK(REGDAT)AND192)=NB THEN 3510
3530 RETURN
3540 :
4000 REM **** COMPROBAR PARACHOQUES ****
4010 IF(PEEK(REGDAT)AND192)=RB THEN
     SS=RT:GOSUB5000:RETURN
4020 IF(PEEK(REGDAT)AND192)=LB THEN
     SS=LF:GOSUB5000:RETURN
4030 RETURN
4040 :
5000 REM **** BUSCAR (SS) ****
5010 DR=FW:DS=8:GOSUB7000:REM MOVER
5020 IF(PEEK(REGDAT)AND192)=NB THEN 5010
5030 RETURN
5040 :
6000 REM **** SONDEAR (WY,SP) ****
6010 IF WY=RT THEN OW=LF
6020 IF WY=LF THEN OW=RT
6030 DR=BW:DS=30:GOSUB7000:REM MOVER
6040 DR=WY:AG=90:GOSUB7500:REM GIRAR
6050 DR=FW:DS=SP:GOSUB7000:REM MOVER
6060 DR=OW:AG=90:GOSUB7500:REM GIRAR
6070 DR=FW:DS=8:GOSUB7000:REM MOVER
6080 IF(PEEK(REGDAT)AND192)=NB THEN 6070
6090 RETURN
6100 :
7000 REM **** MOVER (DR,DS) ****
7010 POKE REGDAT,(PEEK(REGDAT)AND 1)OR DR
7020 PL=PD*DS
7030 FOR I=1 TO PL:GOSUB8000:NEXT I
7040 RETURN
7050 :
7500 REM **** GIRAR (DR,AG) ****
7510 POKE REGDAT,(PEEK(REGDAT)AND 1)OR DR
7520 PL=PA*AG
7530 FOR I=1 TO PL:GOSUB8000:NEXT I
7540 RETURN
7550 :
8000 REM **** IMPULSO ****
8010 POKE REGDAT,PEEK(REGDAT)OR 8
8020 POKE REGDAT,PEEK(REGDAT)AND 247
8030 RETURN
```

Para conocerte mejor

En nuestra serie dedicada al software vertical analizaremos un juego de programas para la "clasificación de la personalidad"

Human Edge Software, una empresa de programación con base en California, ha dado un paso intermedio hacia la inteligencia artificial. Los programas de Human Edge constituyen refinadas herramientas para tomar decisiones comerciales, que trabajan rápidamente a partir de grandes cantidades de información que le proporciona el usuario, evaluando los datos de acuerdo a criterios almacenados y produciendo luego un curso de acción aconsejable. Human Edge es un juego de cuatro programas (*Communication edge, Sales edge, Management edge* y *Negotiation edge*) para máquinas IBM y compatibles. Se afirma que con él se "incrementan las aptitudes profesionales individuales del usuario" en las áreas especificadas en los nombres de cada uno de los programas (comunicaciones, rentas, dirección y negociaciones). Para el Commodore 64, el Apple II y el Macintosh se ha creado una versión reducida, llamada *Mind prober*, que utiliza las mismas técnicas; pero en este capítulo nos centraremos en el juego de cuatro programas.

Se dice que los programas son el producto de más de diez años de desarrollo, suponiendo el trabajo de científicos del comportamiento y expertos en gestión empresarial, e incorporando nuevas técnicas como el análisis de factores humanos, tecnología de sistemas expertos y matemática de teoría de decisiones.

Esta descripción suena bastante técnica y, sumada al costo de los programas, podría intimidar al usuario potencial. No obstante, los programas son fáciles de utilizar y se los puede hacer operar al completo en menos de una hora de autoaprendizaje. Todos ellos se ejecutan a través de menús y están construidos en base a extensos cuestionarios compuestos por una serie de sentencias cuidadosamente redactadas sobre las que se invita al usuario a expresar su conformidad o disconformidad. Las sentencias se dirigen a la evaluación de las características significativas de la personalidad del usuario, así como sus perspectivas de ventas, clientes actuales, subordinados y superiores de la empresa, y cualquier aspecto de las relaciones empresariales que desee investigar el usuario.

El programa evalúa las respuestas y prepara un informe detallado, incluyendo un curso de acción recomendado. Las recomendación puede ser la sugerencia de aproximarse abiertamente a un nuevo cliente, una eficaz estrategia de cierre para una venta difícil o técnicas de negociación a utilizar con empleados o jefes.

Cada programa comienza con un cuestionario de autovaloración que incluye sentencias tales como: "Yo asumo el mando en la mayoría de las reuniones", "Discuto con los demás más que la mayoría", "Soy más bien impulsivo", etc. El usuario decide si la sentencia es una descripción exacta de sí mismo y luego entra su respuesta. La herramienta de autovaloración se ha preparado con gran eficacia, con una considerable superposición de preguntas como medición interna de validez. Por lo tanto, las respuestas del usuario a "Soy más bien impulsivo" y a "A veces actúo sin pensar" se evaluarán una en relación a la otra por razones de coherencia. Completada esta sección, las respuestas se almacenan en disco. Éstas se pueden actualizar y volver a utilizar en cualquier momento.

Después de efectuada la autovaloración, se solicita al usuario que exprese su acuerdo o desacuerdo con una serie de adjetivos relacionados con el objeto de la investigación. Para ayudar al usuario a evaluar a su cliente, su empleado o su jefe, se utilizan palabras tales como hablador, aprensivo, independiente, tenaz, ambicioso, cortés, ostentoso, etc. Al trabajar con esta lista es de gran ayuda tener a mano un buen diccionario de inglés norteamericano para comprender algunos términos, porque el Oxford English Dictionary no reconoce determinados calificativos.

El usuario puede desplazarse por la lista de adjetivos y cambiar sus respuestas en cualquier momento. Al igual que la autovaloración, la lista se guarda en disco, si bien también puede actualizarse según lo dicte la experiencia.

Sólo es necesario completar la autovaloración una vez; luego puede ser recuperada desde el disco y utilizada en relación a cualquiera de las otras lis-

David Higham

tas del "oponente". Se pueden guardar en disco hasta ocho evaluaciones de oponentes. Cuando hay presentes dos listas completas, el programa toma las respuestas y las evalúa, preparando después un informe en el cual se sintetizan las características del usuario y su tema.

El siguiente informe, generado por *Communication edge*, está basado en un usuario real y su oponente (el hijo adolescente del usuario), describiéndose este último como "señor T" (de *test*: prueba). El informe se presenta como si el ordenador le estuviera hablando directamente al usuario:

Para comunicarte con el señor T necesitarás usar tu enfoque flexible y estable de aproximación a la gente. El señor T es una persona muy reservada que prefiere estar sola y tiene muy poca paciencia para mantener charlas y relaciones sociales. Cuando le pidas sus ideas y sus impresiones, espera de él una actitud cínica o sospechosa. Cuando le hables, no supongas por defecto que te está entendiendo. Sé claro, conciso y directo.

Contrastando con tu estilo sereno, el señor T se enfada enseguida e incluso puede parecer enojado antes de que comience la conversación. Podría tratar de imponerte sus propias opiniones. Sé cordial, aunque se porte así. No te extrañes de lo imprevisible de su comportamiento. El señor T puede hablar de forma impulsiva durante un minuto, para, al minuto siguiente, escoger con sumo cuidado cada una de sus palabras. Asume el papel de director del proceso de la conversación. Parafrasea sus comentarios para conseguir claridad y llegar a un acuerdo.

Los padres probablemente reconocerán la descripción de un típico adolescente, si bien *Communication edge* no hace preguntas acerca de la edad del sujeto (sin embargo, sí se tiene en cuenta el sexo).

El vocabulario empleado en los informes les será familiar a quienes lean las columnas de consejos de los periódicos o a quienes hayan participado en tests de personalidad similares. El usuario por lo general se describe en términos amables ("de temperamento sereno, flexible, estable"), mientras que el oponente se ve menos favorecido ("se enfada enseguida, es impredecible, cínico, desconfiado"). Es probable que esto esté diseñado para reforzar la imagen que el usuario tiene de sí mismo y quizá para ajustarse a la forma de pensar de los norteamericanos, en el sentido de considerar los negocios como una guerra, siendo la previsión de ventas o el cliente el enemigo cuya resistencia se debe vencer.

En *Negotiation edge* este enfoque se hace aún más evidente; en el mismo dichas estrategias se recomiendan (impresas en mayúscula) como:

SAQUE PARTIDO DEL CONOCIMIENTO QUE POSEE SOBRE EL SR T
PREPARE AL SR T CON UNAS TEMPRANAS CONCESIONES
ACOSTUMBRE AL SR T A DECIR "SÍ"
ENCUBRA SUS AMENAZAS
EXAGERE SUS PROBLEMAS
SIMULE LA CUANTÍA DE SUS BENEFICIOS
PASE UN BUEN INFORME

Parece ser que quienes desarrollaron los programas asumieron que un usuario sólo tendría uno de los cuatro programas, dado que cada uno de ellos exige que el usuario complete una autovaloración independiente, presentando preguntas similares por un orden ligeramente distinto. A la vista del precio, esta suposición parece razonable; pero puesto que una organización grande probablemente deseará utilizarlos como herramientas para la dirección, hubiera sido más provechoso poder emplear en los cuatro paquetes la misma autovaloración. Pero, dado que los programas son tan parecidos, uno sencillamente puede tomar el menos caro del juego, el *Communication edge*, y adaptar sus informes de modo que se adapten a las diversas situaciones.

Una importante objeción a señalar en los programas, cuando se los va a utilizar como herramientas de dirección, es su incapacidad para aprender a partir de la experiencia, a excepción de la propia actualización que efectúe el usuario de sus valoraciones. Por ejemplo, para éste sería de gran valor poder entrar los resultados de una estrategia propuesta de modo que pudiera modificarla a la luz de la experiencia, especialmente para la evaluación de sujetos de quienes inicialmente se sabe muy poco. Además, muchas de las preguntas son difíciles de responder sobre una base estricta de "estoy de acuerdo o no lo estoy", y no existe evidencia de una estructura arborescente para la formulación de las preguntas, lo que permitiría la ampliación o verificación de las respuestas. Una solución podría ser incluir una opción "no lo sé", que abriría entonces el camino para una pregunta de nivel inferior que se pudiera responder afirmativa o negativamente.

En el análisis final, uno puede creer o no en este tipo de enfoque a las relaciones humanas. Quizá la forma más rentable de utilizarlo sería como ayuda para una preparación concienzuda antes de una entrevista, pero entonces el usuario habrá de tener cuidado en no tomarse demasiado al pie de la letra los consejos; al menos no hasta que los ordenadores se conviertan en máquinas realmente pensantes.

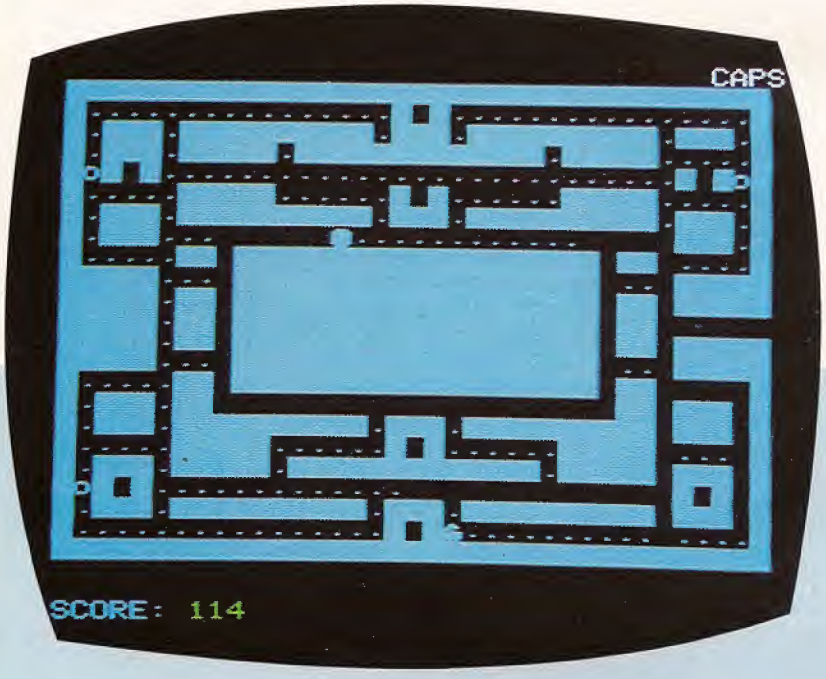
The Human Edge
Juego de cuatro paquetes (que se pueden adquirir por separado) para máquinas MS-DOS y PC-DOS

Distribuidor:
Thorn EMI Computer Software Distributors, 296 Farnborough Road, Farnborough, Hants GU14 7NF, Gran Bretaña
Autores:
Human Edge Software, California
Formato:
Disco

Mind Prober
Para el Commodore 64, el Apple II y el Macintosh
Distribuidor
El mismo
Autores:
Los mismos
Formato:
Disco

Pacman

Presentamos una versión para el Oric de este famoso y pionero juego de laberinto



Este listado ofrece dos particularidades:

1. Hay sólo un fantasma.
2. Al ingerir las "píldoras de energía", el jugador obtiene puntos, pero no puede comerse al fantasma.

Utilice las teclas de control del cursor para desplazarse por la pantalla.

```
 30 RE=0:GOSUB 9000
 40 GOSUB 8000
 50 V%=19:H%=19:GOSUB 7000
 60 PLOT H%,V%," "
 70 A$=KEY$
 80 A=DEEK(783)
 90 REM
 95 H1%=H%:V1%=V%
100 REM
110 IF A=48351 THEN H%=H%-1:M$="'"
120 IF A=48255 THEN H%=H%+1:M$="%"
130 IF A=48319 THEN V%=V%+1:M$="&"
140 IF A=48375 THEN V%=V%-1:M$="$"
150 IF H%<1 THEN H%=38
160 IF H%>38 THEN H%=1

165 PLAY 0,1,1,10
170 IF SCRN(H%,V%)=44 THEN V%=V1%:H%=H1%
    :GOTO 210
180 IF SCRN(H%,V%)=46 THEN SC%=SC%+1:C0%
    =C0%+1
190 IF SCRN(H%,V%)=111 THEN GOSUB 1000
200 IF SCRN(H%,V%)=43 THEN GOTO 5000
210 PLOT H%,V%,M$
220 PLOT 1,25,"PUNTOS:"+STR$(SC%)
230 PLOT X%,Z%,C$
240 Z1%=Z%:X1%=X%
245 IF M1%>3 THEN M1%=0
250 IF M1%=0 THEN Z%=Z%-1
260 IF M1%=1 THEN X%=X%-1
270 IF M1%=2 THEN Z%=Z%+1
280 IF M1%=3 THEN X%=X%+1
281 PLOT H%,V%,"#"
290 IF SCRN(X%,Z%)=44 THEN Z%=Z1%:X%=X1%
    :M1%=INT(RND(1)*4)
300 IF SCRN(X%,Z%)<40 AND SCRN(X%,Z%)>33
    THEN GOTO 5000
310 C$=CHR$(SCRN(X%,Z%))
320 PLOT X%,Z%,"+"
330 IF LV%<1 THEN GOTO 2000
340 IF C0%=T% THEN C0%=0:GOTO 50
350 GOTO 60
1000 SC%=SC%+(INT(RND(1)*20))+20
1010 ZAP
1020 RETURN

2000 CLS
2005 ZAP
2010 PRINT,,,"SE ACABO"
2020 PRINT
2030 PRINT,,,"PUNTOS";SC%
2040 PRINT
2050 PRINT
2060 IF SC%>RE THEN RE=SC%
2070 PRINT,,"MARCADOR ACTUAL:"RE
2080 PRINT
2090 PRINT
2100 PRINT,"PULSAR UNA TECLA PARA JUGAR"
2110 IF KEY$<>"" THEN GOTO 2120
2120 GET K$
2130 GOTO 40
5000 PLOT X%,Z%,"-"
5010 PLOT H%,V%,"#"
5020 PLAY 1,0,1,10
5030 WAIT 15
5040 PLOT H%,V%,"("
5050 PLAY 1,0,1,20
5060 WAIT 20
5070 PLOT H%,V%,")"
5080 PLAY 1,0,1,25
5090 WAIT 30
5100 PLOT H%,V%,"*"
5110 PLAY 1,0,1,30
5120 WAIT 40
5130 PLOT H%,V%," "
5140 PLAY 1,0,1,50
5150 EXPLODE:WAIT 100
5160 C0%=0:V%=12:H%=16:LV%=LV%-1
5170 IF LV%<1 THEN 2000
5180 GOTO 50
7000 CLS:PAPER 0:INK INT (RND(1)*4)+3
7005 PLOT 0,25,6
7006 PING
7010 PRINT"
     ,,,
```

Mejorando lo presente

Acaba de aparecer el RS128, de Memotech, que incorpora facilidades completas de interface

A pesar de ser máquinas tenidas en muy buena estima, la serie de microordenadores Memotech 500 (el MTX500 y MTX512) ha sido ignorada en gran medida por los compradores de ordenadores personales. Características atractivas (tales como gráficos en alta resolución, un ensamblador incorporado, un BASIC sofisticado y un lenguaje exclusivo para tratamiento de textos denominado NODDY) por cierto no han restado mérito a estas máquinas; el fracaso que representa no haber conseguido un gran éxito a nivel popular se puede atribuir a que quedan comprendidas entre dos segmentos diferentes del mercado de ordenadores personales.

Por un lado, su precio tal vez las haga poco asequibles al amante de los juegos, quien puede pensar que las características más sofisticadas no valen el costo más elevado. Por otra parte, el usuario "serio" (Memotech afirma que la serie está dirigida al usuario de pequeña empresa) puede desanimarse por el hecho de que la serie 500 carezca de interfaces incorporadas, lo que permitiría la conexión de unidades de disco. Estas interfaces sí se pusieron a la venta, pero salieron como elementos separados, diseñados para instalarse en un conector marginal situado en el interior de las máquinas. Esto no es demasiado sorprendente tratándose de una empresa que cimentó su nombre a través de la producción de placas accesorias para el ZX81, pero si es posible que haya fracasado en impresionar a los usuarios que deseaban una máquina lista para enchufar y poner en funcionamiento. Memotech parece haber reconocido este problema y ha introducido el RS128, una máquina con interfaces incorporadas.

El aspecto de la máquina

A primera vista, el RS128 parece idéntico a la serie 500; da la impresión de ser elegante y estar un poco por encima del nivel medio del mercado. Al igual que sus medio hermanas, la máquina tiene una carcasa de aluminio en vez de la habitual de plástico, y ello hace que la máquina Memotech sea considerablemente más pesada que la mayoría del resto de micros. Hay un teclado QWERTY estándar y un teclado numérico, que lleva algunas de las instrucciones para el lenguaje de programación de textos NODDY. A la derecha del teclado numérico hay, asimismo, ocho teclas de función programables. Las teclas poseen un tacto excelente y están diseñadas de acuerdo a un elevado estándar profesional.

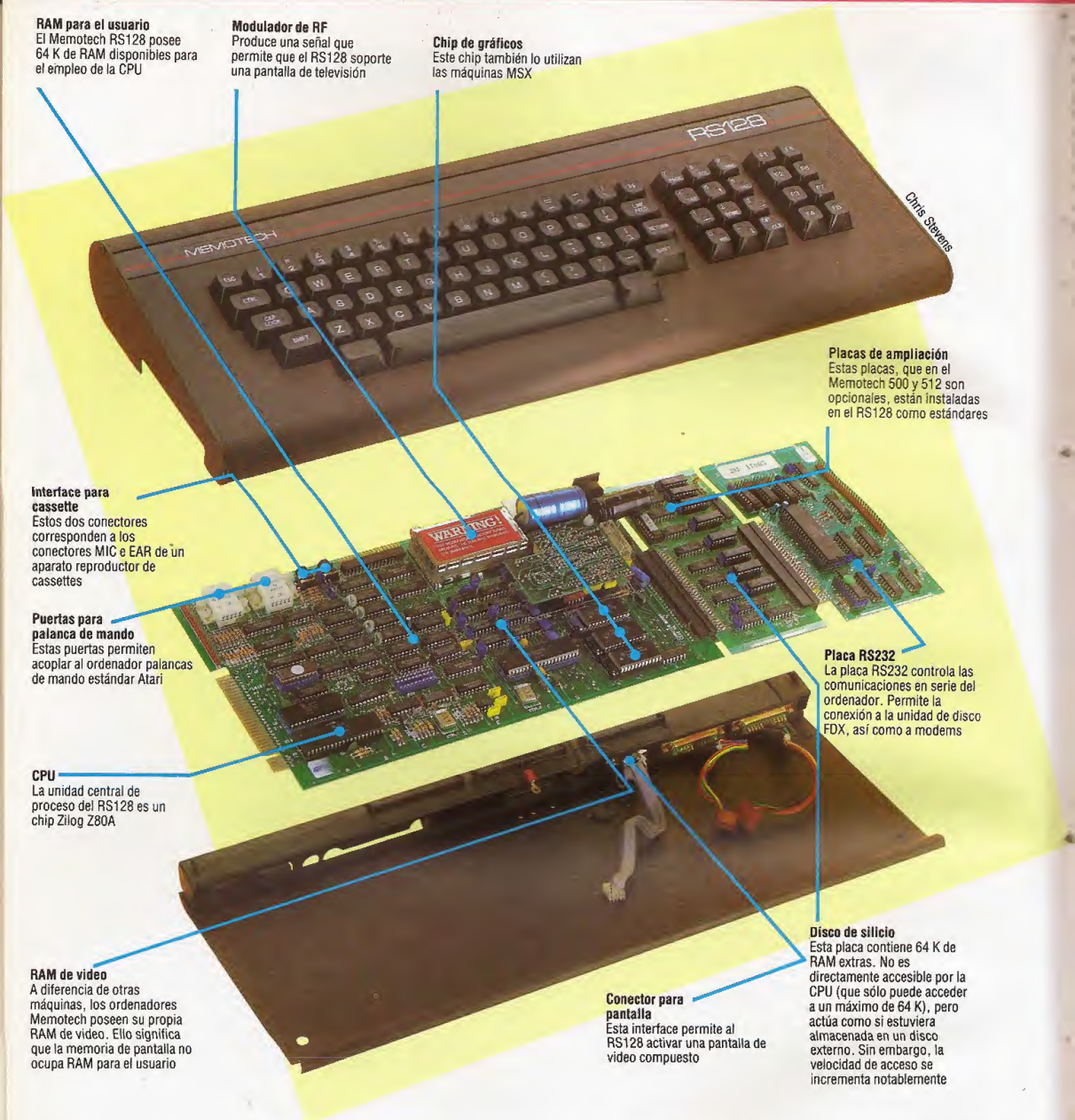
El trazado tiene algunos problemas menores: la tecla Return no es mucho más grande que las teclas normales y al principio los mecanógrafos al tacto tendrán dificultades para localizarla, y la tecla Delete no está en el teclado de máquina de escribir propiamente dicho, sino situada en la zona del teclado numérico. En el rincón superior derecho hay una tecla Backspace pero, a diferencia de la mayor parte de los teclados de ordenador, en los que Backspace actúa también como tecla Delete-left (conocida como "retroceso con eliminación"), en el Memotech es simplemente un cursor-izquierda.

En la parte posterior de la máquina hay numerosas interfaces. Algunas de éstas se suministraban con la serie 500 y otras son adiciones recientes. En el extremo izquierdo hay un par de puertas RS232 que permiten conectar la máquina a unidades de disco flexible FDX. Estas puertas también se pueden utilizar para otros fines, como impresoras en serie y comunicaciones en red. A la derecha de las RS232 hay un enchufe hembra para video compuesto y un enchufe hembra para alta fidelidad (este último permite amplificar el sonido del ordenador a través de un sistema estéreo normal). El conector de potencia y el enchufe hembra RF vienen a continuación, seguidos por una interface para impresora tipo Centronics. La interface para cassette se compone de un par de conectores *microjack*, para EAR y MIC, en el mismo estilo que el Sinclair Spectrum. Por último, hay un par de puertas para palanca de mando tipo Atari de nueve patillas.

Las puertas para interfaces están señaladas con letras blancas, que se pueden leer claramente desde la parte de atrás de la máquina. Esto parece que podría permitir enchufar los periféricos sin tener

Chris Stevens

Modelo perfeccionado
El Memotech RS128 es una versión perfeccionada de la serie MTX500. Este nuevo modelo está dotado de conectores RS232 gemelos, lo que permite que la máquina soporte las unidades de disco flexible FDX. Ello significa que el ordenador es particularmente atractivo para el usuario serio de un micro personal o para el usuario de una pequeña empresa



que inclinarse por arriba para mirar la parte de atrás. Lamentablemente, Memotech ha colocado las puertas en depresiones de la máquina, de modo que el usuario tendrá igualmente que estirarse por arriba para localizar los enchufes.

La pantalla de BASIC, de 24 por 40 caracteres, está dividida en tres secciones, que se aprecian en el momento del encendido. Las 19 filas superiores corresponden a la pantalla principal, donde se visualizan los listados de programas. Debajo está la pantalla EDIT, donde se entran las nuevas líneas. En la parte inferior de la pantalla hay una única línea para la visualización de mensajes de error. Al igual que las máquinas Sinclair, las líneas de programa se modifican mediante el empleo de una instrucción EDIT. Además, el sistema operativo no permite la inserción de una línea en el programa desde la pantalla EDIT si la línea contiene un error de sintaxis.

El BASIC es muy similar al MSX, con instrucciones tales como SOUND, PAPER, INK y CIRCLE. Sin

embargo también contiene algunas útiles instrucciones de las que carece el BASIC MSX. Éstas, en conjunto, están relacionadas con las capacidades para tratamiento de pantalla de la máquina. A modo de ejemplo, la instrucción CSRx,y posicionará el cursor en el punto de la pantalla especificado por las coordenadas (x,y). Una instrucción más potentes es CRVS, que permite que el usuario defina una ventana en cualquier lugar de la pantalla. Dentro de estas ventanas se puede visualizar tanto texto como gráficos.

El lenguaje tiene incorporadas, asimismo, instrucciones para controlar sprites. En este sentido, una instrucción particularmente útil es GENPAT, que permite establecer el patrón del sprite, en vez de tener que colocar el patrón en sentencias de datos. Los gráficos del Memotech los proporciona el chip de video TMS9929A, que es el especificado para las máquinas MSX.

El procesador central de las máquinas Memotech es el Z80 y éste, por supuesto, les permite ejecutar el sistema operativo CP/M. Muchos pequeños fabricantes de ordenadores eligieron el procesador Z80 porque ejecuta CP/M, lo que evita el problema de tener que generar una gran base de software para que los usuarios puedan sacar el máximo partido de un ordenador nuevo. Evidentemente, para aprovechar al máximo el CP/M, el ordenador ha de tener una pantalla de 80 columnas y, aunque es inusual, Memotech proporciona, dentro de la unidad de disco, una placa para uso de 80 columnas. Las unidades, por su parte, son de doble cara y doble densidad, y su velocidad de transferencia al ordenador es de 9 200 baudios.

Con la unidad de disco se suministra un lote de software. Aparte del disco de sistema CP/M, el paquete incluye el procesador de textos *NewWord*, la hoja electrónica *SuperCalc*, *Compact* y *Televideo*, que permiten que las unidades lean discos escritos en otros formatos de disco (según Memotech, ello incluye a los discos IBM) y *Contact*, que hace posible enlazar la segunda puerta RS232 en un sistema de red.

El RS128 posee 128 K de RAM. No obstante, dado que emplea un procesador de 8 bits, sólo es capaz de direccionar 64 K, utilizándose los otros 64 K como un "disco de silicio". Un disco de silicio almacena archivos y programas exactamente de la misma forma en que lo hace un disco flexible, pero como está retenido en chips, es hasta 50 veces más rápido que un disco flexible convencional. La información almacenada en un disco de silicio se transfiere a RAM direccionable cuando así se requiere.

El manual que se proporciona con la máquina es mucho más grande que los que se suministran normalmente con ordenadores personales, si bien ello se debe básicamente a que no ha sido tipografiado, y no a que incluya mucha más información. Sin embargo, Memotech ha incluido todos los detalles técnicos que puede necesitar un usuario, incluyendo diagramas de circuitos y llamadas al sistema operativo.

Al perfeccionar la serie 500 con el RS128, la empresa se ha esforzado tenazmente por producir un ordenador de oficina estándar. Por su precio, la máquina ciertamente entra en competencia con el Sinclair QL, el BBC Micro y el Commodore Plus/4. Sin embargo, queda por ver si generará un volumen de ventas comparable al de sus rivales.

Unidad de disco gemela FDX
Esta unidad permite que el ordenador ejecute el sistema operativo CP/M. Cada una de las unidades de disco de 5 1/4 pulgadas puede almacenar hasta 500 K de información

Ian McKinnell

MEMOTECH RS128

DIMENSIONES
488×202×56 mm

CPU
Z80A operando a 4 MHz

MEMORIA
64 K de RAM y 24 K de ROM

PANTALLA
40×24 en modalidad de textos. Modalidad de texto con gráficos: texto en 32×24 y 256×192 pixels en 16 colores. Existen asimismo facilidades para controlar hasta 32 sprites de forma independiente

INTERFACES
Puertas para cassette (MIC e EAR); interface de E/S; dos puertas para palanca de mando; dos puertas RS232; enchufe para hi-fi; enchufe para video compuesto; enchufe para TV; interface en paralelo

LENGUAJES
BASIC, FORTH, PASCAL

TECLADO
57 teclas tipo máquina de escribir; las teclas F y J están ahuecadas para localización con el dedo. Doce teclas de función en un teclado numérico y ocho teclas de función programables

DOCUMENTACION
El manual que se entrega es el mismo que el de la serie MTX500

VENTAJAS
La adición de las placas RS232 hacen que el RS128 sea una adquisición muy atractiva para el usuario personal "serio" y los usuarios de gestión de pequeñas empresas

DESVENTAJAS
Hay poco software escrito especialmente para la máquina

Software de apoyo

Éstos son algunos de los paquetes de gestión y de juegos que se venden para las máquinas Memotech. En comparación con otras máquinas, el software disponible es bastante limitado, pero el acceso a la bolsa común de software CPM solucionará en gran medida este problema

Alta resolución

Ahora diseñaremos y programaremos las visualizaciones de escenarios especiales para el Commodore 64

El Commodore 64 dispone de facilidades para alta resolución, pero éstas sólo están disponibles para el programador en lenguaje máquina, ya que no se facilita ninguna instrucción en BASIC para el tratamiento de alta resolución, y llevar a cabo los PEEK y POKE correspondientes en BASIC para producir visualizaciones en alta resolución es tan lento que esta aplicación resulta inoperante. En cambio, hemos de adaptar las facilidades, relativamente fáciles de utilizar, que ofrece la máquina.

Se pueden emplear caracteres para gráficos para construir letras grandes u otras visualizaciones mediante la combinación de diferentes caracteres. Los sprites son un método conveniente de introducir formas en alta resolución en la pantalla normal del Commodore. Los caracteres PET se pueden posicionar en la pantalla utilizando ya sea una serie de sentencias PRINT o bien colocando (POKE) el código de carácter correspondiente en la pantalla. Vamos a demostrar ambos procedimientos.

Las líneas 8020-8170 del listado de la pantalla de la palanca de mando comprenden la lectura de los datos para los dos sprites utilizados en esta rutina. El primero, el sprite 0, se define mediante los primeros 63 números del grupo de sentencias DATA comprendidas entre las líneas 8450 y 8497, y representa la puerta para palanca de mando (ilustrada en el diagrama). Los datos de sprites por lo general se sitúan en la parte superior de la zona para programas en BASIC, pero cuando se trata de un programa en BASIC extenso estos datos tienen grandes posibilidades de ser machacados. Una ubicación alternativa es la zona del buffer de la cassette, entre las posiciones 832 y 1022, donde se puede almacenar los datos de tres sprites. Esta rutina almacena sus datos de sprites en esta zona segura.

El sprite 0 sufre una ampliación al doble de su anchura original, para producir la forma final visualizada, colocando a uno el bit 0 del registro de ampliación horizontal en la línea 8170. Observe que todos los registros que controlan los atributos de los sprites, como color, posición y ampliación, están relacionados con la dirección de comienzo para el chip controlador de video (VIC). Recordar que el registro de ampliación horizontal tiene la dirección VIC+29 es mucho más fácil que memorizar su verdadera posición de memoria (53277). Algunos atributos de los sprites requieren un registro entero para cada sprite (p. ej., los registros de las coordenadas X e Y); pero cuando los ocho bits controlan independientemente una función para los ochos sprites disponibles, los atributos se pueden controlar activando y desactivando el bit apropiado en un único registro. El sprite 1 se define mediante los 13 números restantes de las sentencias DATA y representa un objeto a disparar desde la puerta para la palanca de mando.

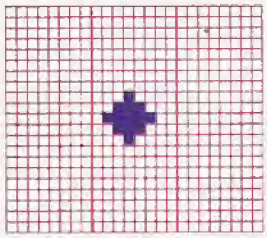
Dado que la parte "sólida" del sprite 1 es pequeña (representa un proyectil), es más rápido y sencillo entrar en dos etapas los 63 bytes de datos que lo definen. Primero, colocando (POKE) 63 ceros en la zona de definición, y luego leyendo (READ) y colocando (POKE) los pocos números que definen la forma. De esta manera podemos omitir la gran cantidad de ceros que, de lo contrario, se requerirían como datos.

Las líneas 8190-8220 se refieren a la construcción de series compuestas por un conjunto de caracteres para gráficos PET. LE$ forma una línea horizontal en toda la anchura de la pantalla, mediante la combinación de 40 caracteres PET especiales de la derecha de la tecla C. DW$ es una serie de caracteres de cursor-abajo. LS$ y RS$ son grupos de diagonales a izquierda y derecha (en frente y a la derecha de las teclas N y M) que se utilizan para formar un patrón de espina de pescado en el primer plano. Este patrón crea en la escena una sensación de profundidad y perspectiva.

La rutina "Disparo" de la línea 8310 elige un punto al azar de la parte inferior de la pantalla y dirige el sprite 1 hacia el mismo, repitiéndose el proceso hasta que el jugador pulsa una tecla. Los

¡Dispárales!
La acción de "disparo" de la puerta para palanca de mando se consigue en el 64 mediante el empleo de un sprite definido como proyectil. Los contornos se consiguen colocando (POKE) ceros en la zona de definición, leyendo (READ) luego las zonas sólidas de sentencias DATA y colocándolas en las posiciones de sprites adecuadas

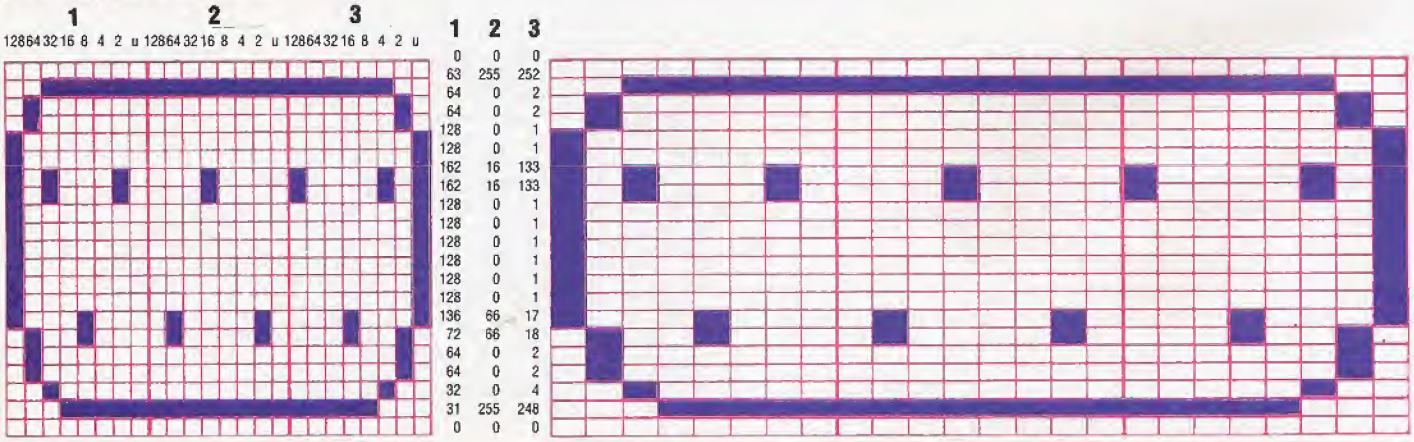
PROYECTIL-SPRITE 1



```
0   16   0
0   56   0
0  124   0
0   56   0
0   16   0
```

Estiramiento de sprites
Los valores listados para el sprite de la puerta para palanca de mando se leen de sentencias DATA y se colocan en las posiciones de sprites apropiadas. Tal como está diseñada, la puerta para palanca de mando está demasiado comprimida, pero se la puede ensanchar horizontalmente (como vemos en la ilustración) cambiando el valor del registro de ampliación horizontal

PUERTA PALANCA-SPRITE 0



Liz Dixon

colores de la pantalla se restauran a los normales, se limpia la pantalla y se apagan los sprites antes de retornar al programa principal. Para utilizar esta subrutina con *Digitaya*, debe insertarse esta línea:

```
3845 GOSUB 8000: REM IMAGEN DE LA PUERTA
     PARA PALANCA DE MANDO
```

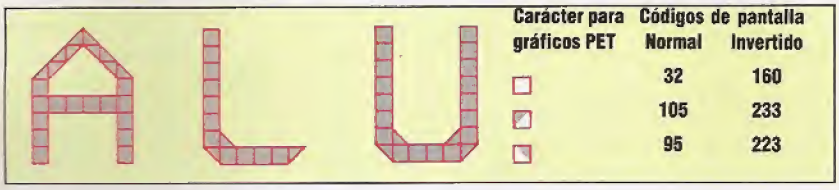
El otro listado proporciona una visualización gráfica para el escenario ALU de *Digitaya* y demuestra distintos métodos de visualizar caracteres en la pantalla. Las líneas de la 7040 y la 7090 leen un cierto número de sentencias DATA y colocan (POKE) los valores directamente en la zona de pantalla. En la posición correspondiente de la zona de color también se le coloca (POKE) el código de color para el carácter. En este ejemplo el código de color es 2, haciendo que los caracteres se visualicen en rojo.

Para conseguir que las grandes letras ALU se desplacen en la pantalla se aplica un truco bastante inusual. La primera línea de códigos de caracteres gráficos que han de conformar las letras ALU se colocan (POKE) en la segunda línea de la pantalla. Entonces se llama a la subrutina de la línea 7680, haciendo que la pantalla se desplace una línea hacia abajo. Luego se puede colocar (POKE) la segunda línea de códigos en la misma zona de pantalla que la primera, y llamar nuevamente a la subrutina. La repetición de este proceso para cada una de las ocho líneas de código hace que las letras ALU parezcan desplazarse desde la parte superior de la pantalla.

Se demuestran otros dos procedimientos para presentar los datos de caracteres en la pantalla. Los caracteres se pueden imprimir (PRINT) directamente, como en las líneas 7130 y 7140, o leer como una serie de datos a imprimir, como es el caso del diseño del signo de interrogación en las líneas 7170 y 7590-7670. Este segundo método permite una sencillez de diseño dentro de las sentencias DATA.

Para utilizar esta rutina añada la siguiente línea:

```
4565 GOSUB 7000:REM IMAGEN ALU
```



Escritura de letras
El escenario de la ALU para *Digitaya* se crea a partir de tres caracteres para gráficos PET en baja resolución, como se puede apreciar. Las grandes letras formadas parecen desplazarse hacia abajo, desde la parte superior de la pantalla hasta su posición de descanso

Liz Dixon

Pantalla ALU

```
7000 REM **** S/R IMAGEN ALU ****
7010 VIC=53248:CS=55296:SC=1024
7020 PRINT CHR$(147):REM LIMPIAR PANTALLA
7030 POKE VIC+32,0:POKE VIC+33,0:REM ESTABLECER PANTALLA/BORDE
7040 CC=0
7050 FOR J=1 TO 8:GOSUB 7680:REM DESPLAZ.
7060 FOR I=47 TO 72
7070 READ A:CC=CC+A:POKE SC+I,A:POKE CS+I,2
7080 NEXT I,J
7090 READ CS:IF CS<>CC THEN PRINT"ERROR SUMA DE CONTROL":STOP
7100 GOSUB 7680:REM DESPLAZ.
7110 PRINTCHR$(158):REM TEXTO AMARILLO
7120 FOR I=1 TO 8:PRINT:NEXT I:REM MOVER HACIA ABAJO
7130 PRINTTAB(9)"AND";SPC(7);"OR";SPC(8);"NOT"
7140 PRINTTAB(10)"0";SPC(9);"0"SPC(9);"0"
7150 PRINTCHR$(28):REM PRUEBA ROJO
7160 REM ** SIGNO INTERROGACION **
7170 FOR I=1 TO 9:READ Q$:PRINTTAB(16)Q$:NEXT I
7180 REM **** ESPERAR TECLA Y REINICIALIZAR ****
7190 GET A$:IF A$="" THEN 7190
7200 POKE VIC+32,14:POKE VIC+33,6:REM PANTALLA/BORDE
7210 PRINTCHR$(154): REM LT TEXTO AZUL
7220 PRINTCHR$(147):REM LIMPIAR PANTALLA
7230 RETURN
7240 REM **** DATOS PANTALLA ****
7250 REM ** FILA 1 **
7260 DATA 160,32,32,32,32,160,32,32,32,32
7270 DATA 95,160,160,160,160,105
7280 DATA 32,32,32,32,95,160,160,160,160,105
7290 REM ** FILA 2 **
7300 DATA 160,32,32,32,32,160,32,32,32,32
7310 DATA 160,223,32,32,32,32,32,32,32,32
7320 DATA 160,223,32,32,233,160
7330 REM ** FILA 3 **
7340 DATA 160,32,32,32,32,160,32,32,32,32
7350 DATA 160,32,32,32,32,32,32,32,32,32
7360 DATA 160,32,32,32,32,160
7370 REM ** FILA 4 **
7380 DATA 160,160,160,160,160,160,32,32,32,32
7390 DATA 160,32,32,32,32,32,32,32,32,32
7400 DATA 160,32,32,32,32,160
7410 REM ** FILA 5 **
7420 DATA 160,32,32,32,32,160,32,32,32,32
7430 DATA 160,32,32,32,32,32,32,32,32,32
7440 DATA 160,32,32,32,32,160
7450 REM ** FILA 6 **
7460 DATA 233,105,32,32,95,223,32,32,32,32
7470 DATA 160,32,32,32,32,32,32,32,32,32
7480 DATA 160,32,32,32,32,160
7490 REM ** FILA 7 **
7500 DATA 32,233,105,95,223,32,32,32,32,32
7510 DATA 160,32,32,32,32,32,32,32,32,32
7520 DATA 160,32,32,32,32,160
7530 REM ** FILA 8 **
7540 DATA 32,32,233,223,32,32,32,32,32,32
7550 DATA 160,32,32,32,32,32,32,32,32,32
7560 DATA 160,32,32,32,32,160
7570 DATA 14463: REM SUMA CONTROL
7580 REM ** DATOS SIGNO INTERROGACION **
7590 DATA "  ???? "
7600 DATA "?      ?"
7610 DATA "?      ?"
7620 DATA "      ??"
7630 DATA "    ?   "
7640 DATA "   ?    "
7650 DATA "  ?     "
7660 DATA "        "
7670 DATA "   •    "
7680 REM **** S/R DESPLAZAR PANTALLA ****
7690 POKE 218,160
7700 PRINTCHR$(19);CHR$(17);CHR$(157);CHR$(148)
7710 RETURN
```

Pant. puerta palanca

```
8000 REM **** S/R IMAGEN PUERTA PALANCA MANDO ****
8010 PRINTCHR$(147):REM LIMPIAR PANTALLA
8020 S0=832:S1=S0+64:REM DIR COMIENZO DATOS SPRITE
8030 CC=0:VIC=53248:REM COMIENZO DE CHIP VIC
8040 FOR I=S0 TO S0+62:READ A:CC=CC+A:POKE I,A:NEXT I
8050 FOR I=S1 TO S1+62:POKE I,0:NEXT I
8060 FOR I=S1+25 TO S1+37:READ A:CC=CC+A:POKE I,A:NEXT I
8065 READ CS:IF CS<>CC THEN PRINT"ERROR SUMA DE CONTROL":STOP
8070 POKEVIC+33,0:POKEVIC+32,0:REM ESTABLECER PANTALLA/BORDE
8080 REM ** ESTABLECER PUNTEROS SPRITES **
8090 POKE 2040,S0/64:POKE 2041,S1/64
8100 REM ** ESTABLECER REGS CONTROL SPRITES VIC **
8110 POKE VIC+39,7:REM ESTABLECER COLOR SPRITE 0
8120 POKE VIC+40,7:REM ESTABLECER COLOR SPRITE 1
8130 POKE VIC,65:REM ESTABLECER COORD X SPRITE 0
8140 POKE VIC+1,70:REM ESTABLECER COORD Y SPRITE 0
8150 POKE VIC+2,74:REM ESTABLECER COORD X SPRITE 1
8160 POKE VIC+3,70:REM ESTABLECER COORD Y SPRITE 1
8170 POKE VIC+29,1:REM AMPLIAR HORIZ SPRITE 0
8180 LE$="":FOR I=1 TO 40:LE$=LE$+CHR$(195):NEXT I
8200 DW$="":FOR I=1 TO 25:DW$=DW$+CHR$(17):NEXT I
8210 LS$="":FOR I=1 TO 11:LS$=LS$+CHR$(206)+"  ":NEXT I
8220 RS$="":FOR I=1 TO 11:RS$=RS$+CHR$(205)+"  ":NEXT I
8230 PRINTCHR$(155):REM TEXTO AMARILLO
8240 PRINTCHR$(19):PRINT TAB(2)"PUERTA PALANCA MANDO"
8250 PRINTCHR$(154):REM TEXTO AZUL CLARO
8260 PRINTCHR$(19);LEFT$(DW$,17);LE$
8270 PRINT CHR$(145);
8280 PT$=LEFT$(LS$,19)+LEFT$(RS$,21)
8290 PT$=PT$+RIGHT$(LS$,19)+RIGHT$(RS$,21)
8300 FOR I=1 TO 3:PRINT PT$;:NEXT I
8305 POKE VIC+21,3:REM ENCENDER SPRITES 0 & 1
8310 REM ** DISPARAR **
8320 YB=240:Y1=70
8330 X1=74:X=INT(RND(1)*150)+24
8340 G=2*(X-X1)/(YB-Y1)
8350 FOR Y=Y1 TO YB STEP 2
8360 X1=X1+G:POKE VIC+2,X1:POKE VIC+3,Y
8370 NEXT Y
8380 GET A$:IF A$="" THEN 8330
8390 REM ** RESTAURAR PANTALLA **
8400 POKE VIC+21,0:REM APAGAR SPRITES
8410 POKE VIC+32,14:POKEVIC+33,6:REM RESTAURAR PANTALLA/BORDE
8420 PRINT CHR$(147):REM LIMPIAR PANTALLA
8430 RETURN
8440 REM **** DATOS SPRITE ****
8450 DATA 0,0,0,63,255,252,64,0,2,64,0,2
8460 DATA 128,0,1,128,0,1,162,16,133,162,16,133
8470 DATA 128,0,1,128,0,1,128,0,1,128,0,1
8480 DATA 129,0,1,128,0,1,136,66,17,72,66,18
8490 DATA 64,0,2,64,0,2,32,0,4,31,255,248
8495 DATA 0,0,0,16,0,0,56,0,0,124,0,0,56,0,0,16
8497 DATA 3701:REM SUMA DE CONTROL
```

Calcular factoriales

El LOGO es un lenguaje ideal para explorar las matemáticas. A partir de procedimientos sencillos se van realizando tareas más complejas

¿De cuántas formas puede uno acomodar a cuatro personas en cuatro sillas alrededor de una mesa? La primera persona se puede sentar en cualquiera de las cuatro sillas, pero después de haberse sentado, para la segunda persona sólo quedan tres opciones, luego quedan dos para la tercera, y a la última persona sólo le queda un lugar. Por lo tanto, la cantidad total de arreglos diferentes es $4\times3\times2\times1$. Esto por lo general se escribe como 4! y se lee "4 factorial". Los factoriales se encuentran con frecuencia en problemas matemáticos que entrañen variaciones, combinaciones y probabilidades.

Es fácil escribir una definición recursiva para calcular factoriales. En primer lugar, debemos observar que el factorial de 0 se define como 1. El factorial de cualquier número positivo distinto de cero (p. ej., x) es el factorial de $x-1$ multiplicado por x. Traduciendo esto en un programa obtenemos:

```
TO FACTORIAL :X
  IF :X=0 THEN OUTPUT 1
  OUTPUT (FACTORIAL :X-1)*:X
END
```

Para probarlo, digite PRINT FACTORIAL 6; el resultado será 720.

Este procedimiento funciona bien hasta el número 12, pero, a partir de éste, los números se vuelven demasiado grandes como para que el ordenador los pueda retener como enteros. En el Commodore 64, por ejemplo, PRINT FACTORIAL 13 dio 6.22702E9, es decir, 6,22702 veces 10^9. Esto es poco satisfactorio, puesto que se han perdido los cuatro últimos dígitos. Existen muchas razones (incluyendo la pura curiosidad) por las cuales podríamos desear conocer cuáles son estos dígitos restantes. Lo primero que hemos de hacer, por lo tanto, es ampliar las capacidades aritméticas del LOGO de modo que pueda calcular con una precisión mayor de siete cifras.

Para simplificar el asunto, sólo vamos a considerar los enteros positivos. Representaremos a los enteros como listas (de modo que representaremos 1.234.567 como [1 2 3 4 5 6 7]). Los dos procedimientos siguientes realizarán la suma de este tipo de números. Probemos con PRINT SUMALARGA[1 2 3] [5 6 9]; el resultado será [6 9 2]:

```
TO SUMALARGA :X :Y
  OUTPUT SUMALARGA1 :X :Y 0
END

TO SUMALARGA1 :X :Y :LLEVO
  IF(ALLOF (EMPTY?:X)(EMPTY?:Y)(:LLEVO
  =0))THEN OUTPUT[]
  TEST EMPTY? :Y
  IFTRUE IF :LLEVO=0 THEN OUTPUT :X ELSE
  OUTPUT SUMALARGA1 :X [1] 0
  TEST EMPTY? :X
  IFTRUE IF :LLEVO=0 THEN OUTPUT :Y ELSE
  OUTPUT SUMALARGA1 [1]:Y 0
  MAKE "SUMA(LAST :X)+(LAST :Y)+:LLEVO
  OUTPUT LPUT REMAINDER :SUMA 10
  SUMALARGA1·BUTLAST :X
  BUTLAST :Y QUOTIENT :SUMA 10
END
```

Estos procedimientos funcionan de forma muy similar a la que utilizaríamos si hiciéramos las sumas en papel, sumando desde la izquierda e incorporando cualquier número arrastrado desde la columna anterior.

La resta es un proceso similar. No obstante, hemos incluido una rutina para suprimir de la respuesta los ceros no significativos, de modo que no acabemos con resultados tales como [0 0 0 7 8].

```
T RESTALARGA :X :Y
  OUTPUT QUITARCEROS RESTALARGA1 :X:Y 0
END

TO RESTALARGA1:X :Y :PIDO
  IF(ALLOF (EMPTY?:X)(EMPTY?:Y)(:PIDO=
  0))THEN OUTPUT [0]
  TEST EMPTY? :Y
  IFTRUE IF :PIDO=0 THEN OUTPUT :X
  ELSE OUTPUT RESTALARGA1 :X[1]0
  IF EMPTY? :X THEN PRINT [LO SIENTO,NO
  PUEDO MANIPULAR RESULTADOS
  NEGATIVOS] TOPLEVEL
  MAKE "DIFF(LAST :X)-(LAST :Y) - :PIDO
  IF:DIFF<0 THEN OUTPUT LPUT (10+:DIFF)
  RESTALARGA1 BUTLAST :X BUTLAST :Y 1
  OUTPUT LPUT :DIFF RESTALARGA1 BUTLAST
  :X BUTLAST :Y 0
END

TO QUITARCEROS :X
  IF EMPTY? :X THEN OUTPUT [0]
  IF NOT ((FIRST :X)=0)THEN OUTPUT :X
  OUTPUT QUITARCEROS BUTFIRST :X
END
```

La multiplicación larga es algo más complicada. La implementaremos utilizando la técnica que normalmente se enseña en las escuelas. Por ejemplo, vamos a suponer que queremos multiplicar 123 por 338. El problema se divide en tres partes: primero multiplicamos 123 por 8; luego multiplicamos 123 por 330; y, por último, sumamos los dos resultados entre sí. Este método se basa en el hecho de que la segunda etapa se puede dividir en dos subetapas: primero se multiplica 123 por 33, y luego se coloca un cero al final del resultado. Multiplicar un número por 33 evidentemente implica el uso de la recursión. El procedimiento MULTLARGA controla esta estrategia general:

0! 1
1! 1
2! 2
3! 6
4! 24
5! 120
6! 720
7! 5,040
8! 40,320
9! 362,880
10! 3,628,800
11! 39,916,800
12! 479,001,600
13! 6,227,020,800
14! 87,178,291,200
15! 1,307,674,368,000
16! 20,922,789,888,000
17! 355,687,428,096,000
18! 6,402,373,705,728,000
19! 121,645,100,408,832,000
20! 2,432,902,008,176,640,000

Factor explosivo

Los valores factoriales aumentan con pasmosa rapidez, como podemos apreciar aquí. Debido a que los valores se vuelven tan grandes, la mayoría de los ordenadores y calculadoras representarán los factoriales de números mayores de 12 en notación exponencial. Por lo tanto, el factorial de 12 se dará como 4.79E8, o 4.79×10^8. Se incrementa la exactitud si se reflejan todos los dígitos significativos

```
TO MULTLARGA :X :Y
  IF EMPTY? BUTLAST :Y THEN OUTPUT
  MULTLARGA1 :X LAST :Y 0
  OUTPUT SUMALARGA(MULTLARGA1 :X
  (LAST :Y)0)
  (LPUT "0 MULTLARGA :X BUTLAST :Y)
END
```

Los detalles que supone la multiplicación de una línea por un único dígito los lleva a cabo MULTLARGA1:

```
TO MULTLARGA1 :X :NUM :LLEVO
  TEST EMPTY? :X
  IFTRUE IF :LLEVO=0 THEN OUTPUT [] ELSE
  OUTPUT (LIST :LLEVO)
  MAKE "PROD (LAST :X)*:NUM+ :LLEVO
  OUTPUT LPUT REMAINDER :PROD 10
  MULTLARGA1 BUTLAST :X :NUM QUOTIENT
  :PROD 10
END
```

Para el cálculo de factoriales no necesitamos procedimientos que efectúen la división, pero usted puede ampliar el sistema para que también cubra la división.

Ahora contamos con un conjunto de primitivas para llevar a cabo aritmética con cualquier grado de precisión. La única limitación respecto al tamaño de los números que se pueden manipular es el espacio total de memoria disponible para el programa.

Introducción de modificaciones

Ahora podemos modificar nuestro programa factorial original para utilizar la nueva forma de multiplicación larga.

```
TO FACT :X
  IF FIRST :X=0 THEN OUTPUT [1]
  OUTPUT MULTLARGA (FACT RESTALARGA
  :X [1]):X
END
```

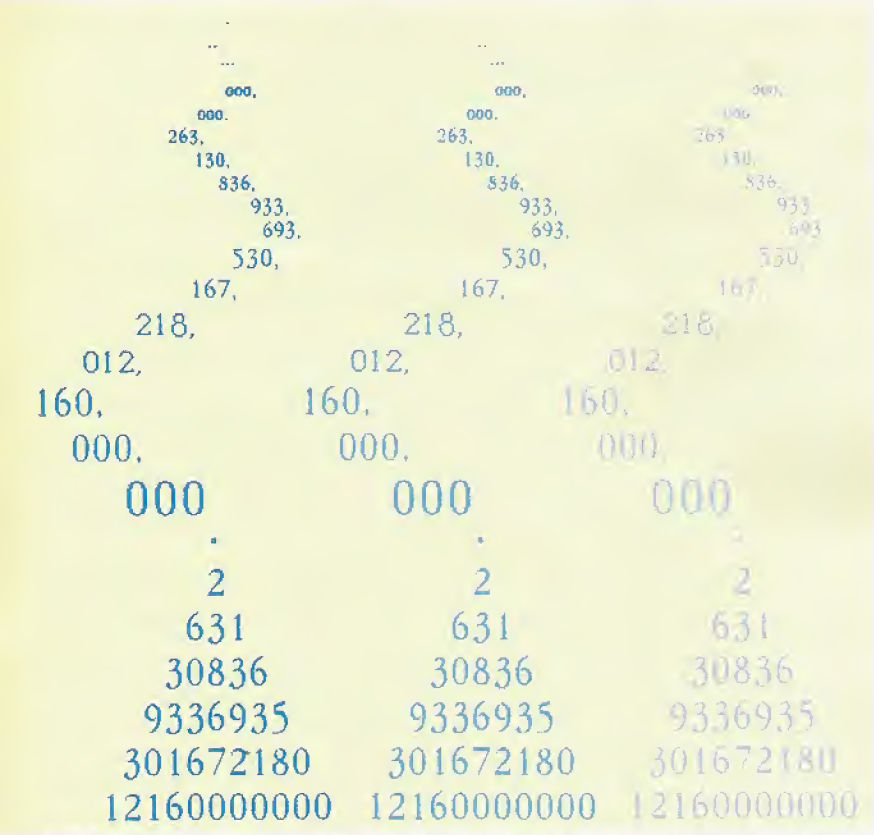
Para probarlo digite FACT[1 3]; obtendrá [6 2 2 7 0 2 0 8 0 0]. Sin embargo, existen problemas. El proceso de cálculo es lento y (en el Commodore 64) el factorial más grande que obtuvimos antes de quedarnos sin memoria fue 34!, que tiene 39 dígitos (y calcularlo llevó algún tiempo).

La expresión de números grandes en forma de listas puede parecer inusual, pero podemos modificar el programa para que supere este problema traduciendo una y otra vez nuestra notación normal a la forma de lista. Empleamos dos procedimientos (EXPLOSION e IMPLOSION) para hacerlo.

EXPLOSION 123 produce [1 2 3] e IMPLOSION [1 2 3] produce 123.

```
TO EXPLOSION :X
  IF EMPTY? :X THEN OUTPUT []
  OUTPUT (SENTENCE FIRST :X EXPLOSION
  BUTFIRST :X)
END

TO IMPLOSION :X
  IF EMPTY? :X THEN OUTPUT"
  OUTPUT (WORD FIRST :X IMPLOSION
  BUTFIRST :X)
END
```

Ian McKinnell



Árbol de números
Los árboles factoriales se generan utilizando el dígito de más a la izquierda de un valor factorial como la copa del árbol. Los dígitos subsiguientes se extraen del valor real, de izquierda a derecha, en grupos cuyos tamaños van aumentando ligeramente. Los grupos se colocan debajo del dígito que constituye la piedra angular de una forma de árbol, y en posición simétrica respecto al mismo. Este diagrama representa el factorial de 32. El valor se puede leer más fácilmente cuando los números se disponen en grupos de tres, como podemos observar

Estos procedimientos se basan en el hecho de que en LOGO los números se tratan como palabras. Utilizándolas, podemos ahora definir un procedimien to, F:

```
TO F :X
  PRINT IMPLOSION FACT EXPLOSION :X
END
```

que calculará el factorial de 13 en respuesta a la entrada: F13.

El resultado de este cálculo (6227020800) es un poco difícil de leer como tal. Es más normal insertar puntos (6.227.020.800), que hacen que resulte más fácil de comprender. Los siguientes procedimientos dividen la palabra en grupos de tres dígitos e insertan puntos.

```
TO AÑADIRPUNTOS :X
  IF ((CONTADOR :X)<4) THEN OUTPUT:X
  OUTPUT(WORD AÑADIRPUNTOS
  CADATRES :X", ULTIMOSTRES :X)
END

TO CADATRES :X
  OUTPUT BUTLAST BUTLAST BUTLAST :X
END

TO ULTIMOSTRES :X
  OUTPUT (WORD (LAST BUTLAST BUTLAST :X)
  (LAST BUTLAST :X) (LAST :X))
END
```

Debemos, asimismo, modificar F para que incorpore estos procedimientos:

```
TO F:X
  PRINT AÑADIRPUNTOS IMPLOSION FACT
  EXPLOSION :X
END
```

La utilización de F para imprimir los primeros 20 factoriales nos da una idea de la rapidez con que crecen en tamaño los factoriales (los resultados se pueden apreciar en la tabla que ofrecemos en la página anterior).

Habiendo obtenido los factoriales de una gama de números, podemos comenzar a "jugar" con nuestros resultados. Un matemático norteamericano, por ejemplo, en una ocasión tuvo la brillante idea de imprimir números factoriales grandes en forma de árboles en las tarjetas de Navidad que les envió a sus amigos.

No son muchos los factoriales que poseen el número adecuado de dígitos para ser impresos como árboles, pero los siguientes procedimientos funcionarán si ello fuera posible:

```
TO ARBOL :L
  ARBOL1 1 :L
END

TO ARBOL1 :NUM :L
  IF EMPTY? :L THEN STOP
  REPEAT ROUND(20 – :NUM/2) [PRINT1
  ESPACIO] IMPRESIONLINEA :NUM :L
  ARBOL1 :NUM+2 PODAR :NUM:L
END

TO ESPACIO
  OUTPUT CHAR 32
END

TO IMPRESIONLINEA :NUM :L
  IF :NUM=0 THEN PRINT "STOP
  PRINT1 FIRST :L
  IMPRESIONLINEA :NUM – 1 BUTFIRST :L
END

TO PODAR :NUM :L
  IF :NUM=0 THEN OUTPUT :L
  OUTPUT PODAR :NUM – 1 BUTFIRST :L
END
```

Nuevamente debemos modificar nuestro procedimiento controlador:

```
TO F :X
  ARBOL IMPLOSION FACT EXPLOSION :X
END
```

El diagrama ilustra 32! escrito en forma de árbol. Si tiene intención de explorar más estos árboles factoriales, quizá le interesará saber que hay sólo tres números menores de 32 cuyos factoriales se pueden escribir como árboles. De los factoriales mayores adecuados el siguiente es 59!

Complementos al LOGO

En todas las versiones LCSI, utilice:

EMPTYP por EMPTY?
AND por ALLOF
TYPE por PRINT1

Existe, asimismo, una sintaxis diferente para IF. P. ej.:

IF :LLEVO=0[OUTPUT[]][OUTPUT (LIST:LLEVO)]

En lugar de QUOTIENT :X :Y emplee DIV :X :Y en el Spectrum y ROUND (:X/:Y) en el Atari.

En el Atari utilice SE por SENTENCE.

El toque FX del BBC Micro

Volvamos a analizar el sistema operativo del BBC Micro con mayor riqueza de detalles. Fijemos nuestra atención en las llamadas OSBYTE, que ofrecen una conveniente manera de acceder a muchas de las funciones del OS de esta máquina

En un primer momento, al estudiar cómo se accede al sistema operativo del BBC hablamos de un grupo de llamadas conocidas genéricamente como OSBYTE. Eran las que nos permitían modificar el comportamiento de varias partes del sistema operativo. Por ejemplo, la llamada OSBYTE *FX4,1 permite cambiar la manera en que el ordenador responde al pulsar una de las teclas del cursor en un teclado del BBC.

Si se piensa que el número de estas llamadas supera el centenar en la versión 1.2 del sistema operativo, no sorprende tanto el hecho de que ofrezcan una manera conveniente de acceder a muchas de las funciones del OS de dicho ordenador. Ya vimos que el empleo de las llamadas indirectas al sistema operativo nos protege contra posibles cambios en la configuración del software y del hardware de la máquina; la OSBYTE nos proporciona el método más importante de empleo de las rutinas del sistema operativo.

Antes de avanzar en el examen de OSBYTE, hay que tener en cuenta que con una máquina provista de una vieja versión 0.1 varias de las llamadas OSBYTE de las que aquí hacemos referencia no están soportadas.

Se comprueba el tipo de versión de una máquina BBC digitando sencillamente *HELP y RETURN.

Veamos primero cómo se emplea OSBYTE desde el BASIC y desde el lenguaje máquina. Como la mayoría de las llamadas del sistema operativo del BBC, también OSBYTE está vectorizada. Su vector se encuentra en las direcciones &20A y &20B.

Más abajo mostramos los modos de realizar una llamada OSBYTE. Todos ellos ejecutan la llamada OSBYTE antes mencionada, *FX4,1:

Desde BASIC usando *FX	Desde BASIC por medio de USR	En código máquina
*FX4,1	A%=4:X%=1:D%=USR(&FFF4)	LDA #4 LDX #1 JSR &FFF4

De estos tres ejemplos aparece claro que la dirección a la que hacemos las llamadas a las rutinas OSBYTE es la &FFF4, y los parámetros se pasan en la llamada OSBYTE en los registros A, X e Y del procesador 6502. Cuando se asigna un valor a A%, a X% y a Y% en BASIC y posteriormente se llama a una rutina del lenguaje máquina por medio de USR o CALL desde el BASIC, el programa en código máquina llamado será introducido con los registros A, X e Y del procesador conteniendo los valores de las variables A%, X% e Y%, respectivamente.

El empleo de USR en el segundo ejemplo nos permite obtener un resultado del programa en código máquina y devolverlo a una variable en BASIC. Lo cual es práctico con algunas llamadas OSBYTE, ya que éstas pueden devolver información al BASIC. De ello nos ocuparemos más adelante.

En el tercer ejemplo, empleamos un breve programa en código máquina para hacer una llamada OSBYTE; se trata de una sencilla carga de los registros necesarios con los valores adecuados y de la llamada OSBYTE a su dirección de llamada, la &FFF4. Estos ejemplos ilustran también la correspondencia existente entre los parámetros pasados al sistema operativo con la llamada FX (o sea, 4 y 1) y los parámetros pasados a los registros A, X e Y cuando llamamos las rutinas OSBYTE por medio del lenguaje máquina o la llamada USR.

Independientemente del modo empleado para llamar a la rutina OSBYTE, el contenido del registro A especificará siempre cuál de las muchas rutinas es la elegida. Los registros X e Y se utilizan entonces para pasar parámetros a la rutina mencionada del OSBYTE.

Algunas llamadas a OSBYTE no necesitan parámetros; otras sólo necesitan un parámetro que se pasará en el registro X; y las menos necesitan dos, pasados en los registros X e Y.

He aquí algunos ejemplos de estas llamadas a OSBYTE:

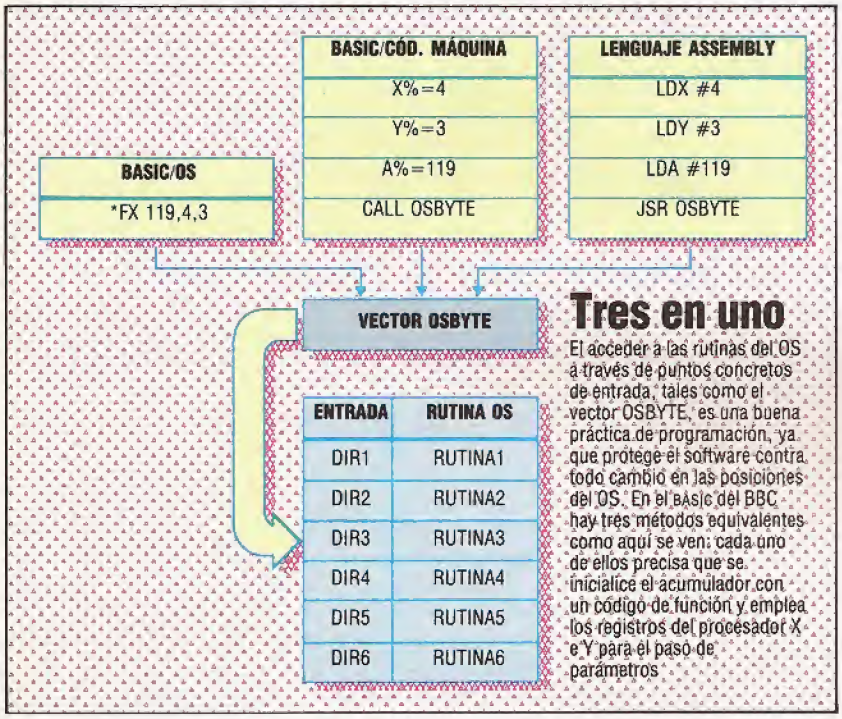
Sin parámetros	Un parámetro	Dos parámetros
*FX0	*FX4,1	*FX151,96,200

Hay que hacer algunas observaciones sobre cómo se llaman las rutinas OSBYTE por medio de *FX. La primera es que OSBYTE desconoce completamente las variables en BASIC, tales como las instrucciones *.

Si ejecutamos la codificación escrita más abajo obtendremos el siguiente mensaje de error Bad Command:

```
a=4:b=1
*FX a,b
```

Sin embargo, para obviar este problema se puede pasar la instrucción FX al OS por medio de OSCLI. Así:



Liz Dixon

Tres en uno

El acceder a las rutinas del OS a través de puntos concretos de entrada, tales como el vector OSBYTE, es una buena práctica de programación, ya que protege el software contra todo cambio en las posiciones del OS. En el BASIC del BBC hay tres métodos equivalentes como aquí se ven: cada uno de ellos precisa que se inicialice el acumulador con un código de función y emplea los registros del procesador X e Y para el paso de parámetros.

```
10 DIM C 100
20 a=4
30 b=1
40 $C="*FX"+STR$a+","+STR$b
50 X%=C MOD 256
60 Y%=C DIV 256
70 CALL &FFF7
```

La segunda observación sobre las llamadas FX se refiere a que no se puede poner nada más en una línea de programa después de ellas.

Veamos:

```
*FX 4,1:PRINT "Oooops!"
```

Esta línea volverá a darnos el mensaje Bad Command: el sistema operativo considera los dos puntos y la instrucción PRINT como parte de la instrucción FX.

Ambos problemas se deben al hecho de que las llamadas *FX se pasan a través del intérprete de línea de instrucciones (CLI: *command line interpreter*) y no a través del intérprete BASIC, siendo el caso que CLI desconoce la manera de valorar las variables BASIC y de tratar líneas con varias instrucciones.

Como vimos, se pueden pasar parámetros al OSBYTE con los registros X e Y, y es posible también releer valores de algunas de las variables del sistema empleadas por el sistema operativo. Esto se hace con el empleo de la llamada USR o de una rutina en código máquina, como ya sabemos. Quizá usted encuentre el método del código algo más fácil de usar en el caso de que le interesen los resultados obtenidos del sistema operativo, ya que el valor obtenido con la llamada USR ha de ser decodificado para conseguir varios bits de información de ella.

Los parámetros son devueltos al BASIC con los registros X e Y, y también en algunas llamadas el flag de arrastre se utiliza para indicar una condición de error.

Obviamente los resultados que se devuelven de este modo dependerán de la llamada, es decir, del valor pasado al OSBYTE con el registro A. No todas las llamadas OSBYTE devuelven resultados al BASIC. De todas formas, muchas de ellas nos facilitan una útil información sobre el sistema operativo.

Información retornada

Son dos los tipos de información que puede retornar una llamada OSBYTE. El primero es el de los datos leídos de alguna parte del sistema, tal como la puerta para el usuario, el procesador de la voz, o las variables del sistema. Las llamadas OSBYTE que devuelven este tipo de información son conocidas como llamadas *de sólo lectura*. Un ejemplo típico de su empleo es la llamada OSBYTE con A=129. Esta llamada se utiliza en BASIC para implementar la función INKEY ().

Los registros X e Y deben establecerse para pasar el retardo temporal adecuado al sistema operativo. El registro X retiene el byte inferior de dicho retardo (en centisegundos) y el registro Y el byte superior. Así, para que esta llamada genere una espera de hasta un segundo hasta la pulsación de una tecla, el fragmento de código máquina que escribimos más abajo es válido. Los registros X e Y devuelven los valores; si el flag de arrastre está a cero, y el registro Y contiene asimismo un valor de 0, significa que la salida fue provocada por la pulsación de una tecla. El valor ASCII de esa tecla se encontrará en el registro X. Si Y contiene 255 y C está a 1, es señal de que no se pulsó tecla alguna en el período de tiempo especificado. Si C está a uno e Y contiene 27, indica que se pulsó la tecla Escape.

El siguiente fragmento en lenguaje máquina ilustra la manera de realizar esta llamada OSBYTE. Si el flag de arrastre está activado, a la vuelta de la rutina se hace una bifurcación a una rutina ulterior de tratamiento.

Esta rutina puede comprobar el valor contenido en el registro Y para saber si se pulsó una tecla o no, o bien se pulsó Escape.

```
1000    LDA     #129    /establece OSBYTE
1010    LDX     #100    /parámetros
1020    LDY     #0
1030    JSR     &FFF4   /realiza la llamada OSBYTE
1040    BCS     error
1050    RTS
1060    .error          /código para tratar un
                        error
```

Otras llamadas OSBYTE, especialmente las que dan un valor a A entre 166 y 255, son llamadas de lectura y escritura a la vez, permitiéndonos leer o bien escribir ciertas variables del sistema en el sistema operativo.

Puede que usted empiece a preguntarse cómo sabe una llamada OSBYTE si le piden una operación de lectura o de escritura. Muy sencillo.

Para escribir un valor con una llamada OSBYTE, la llamada se realizará conteniendo el registro X el valor que deseamos que tal llamada escriba mientras el registro Y está puesto a 0.

Para leer un valor de una de estas variables del sistema, se pone X a 0 e Y contendrá 225. Después se hace la llamada.

Si va a haber una devolución, su valor se pondrá en los registros X e Y.

Empleo de las llamadas OSBYTE

Las llamadas OSBYTE son "cabos furrieles" del sistema operativo, que intervienen en un gran número de rutinas del sistema operativo. Los sistemas de ficheros, el teclado, Econet, las teclas Break y Escape, se ven afectados en mayor o menor medida por las llamadas OSBYTE.

El número de estas llamadas disponibles nos impide examinarlas una por una; aun así trataremos aquí las más útiles y menos estudiadas en el manual del usuario del BBC Micro.

Teclas de función: La orden *FX18 no tiene parámetros, pero es muy útil. Borra de la memoria las definiciones en curso de teclas de función, lo que resulta práctico a la hora de definir varias veces en el programa una tecla de función.

De *FX225 a *FX228: Si no se programó una tecla de función con una serie, se pueden utilizar estas llamadas para obligar a las teclas de función rojas a que nos den el valor ASCII. Por ejemplo, *FX225,n hará que la tecla de función f0 nos dé el código ASCII de n al pulsarla, y la f1 nos dará el de (n+1), etc. La *FX226 hace lo mismo cuando las teclas se pulsan juntamente con la techa Shift. La *FX227 sirve cuando se pulsan las teclas junto con la tecla Control, y la *FX228 lo hace cuando se pulsan a la vez Shift y Control además de una de función.

Funciones video y manejadores de VDU: Gran parte de la tarea de control de la visualización en el BBC Micro se realiza mediante la escritura de valores para los manejadores de VDU. Aun así, un par de llamadas OSBYTE vienen en nuestra ayuda en este contexto.

*FX19: Aunque no tenga parámetros, su utilidad se evidencia al programar gráficos en movimiento. Una vez ejecutada esta instrucción, el ordenador espera hasta que la siguiente "ventana" de visualización se haya escrito. Esto significa que el movimiento de los gráficos es menos "arbitrario". Ha de hacerse la llamada siempre que se necesite una pausa. Dado que la visualización se refresca 50 veces por segundo, es posible su empleo para generar retardos temporales o proporcionar interrupciones a la CPU.

*FX218: Es una llamada OSBYTE de lectura/escritura que nos informa de la longitud de la "cola" VDU (unidad de representación visual). Ya hemos explicado cómo algunos códigos VDU, cuando se envían a través de manejadores VDU, esperan que les sigan otros códigos. El número de bytes que aún esperan los manejadores de VDU en cualquier momento es el número de bytes de la cola VDU. De manera comprensiva, esta llamada se usa mejor para volver a leer información, y el resultado se retornará en el registro X.

*FX20: Esta llamada nos permite redefinir los caracteres en ASCII en el intervalo del 32 al 255, en vez del intervalo habitual y limitado de los caracteres definidos por el usuario. Así podemos redefinir, en Modos del 0 al 6, los caracteres a los que se accede desde el teclado normal. Para redefinir los caracteres en códigos ASCII dentro del intervalo del 32 al 128 debemos reservar memoria en nuestro espacio de trabajo del BASIC poniendo PAGE a un valor más alto que de costumbre. La redefinición de estos caracteres se hace con la llamada VDU 23. Los caracteres se redefinen en bloques de 32, y cada bloque exige 256 bytes de memoria. El único parámetro empleado se pasa al registro X. El manual del usuario proporciona más detalles sobre esta técnica.

Sonido: *FX210,n es una llamada útil para esos juegos generosos en ruidos "destruyeovnis". Permite diluir los efectos sonoros. El sonido se oye normalmente con *FX210,0 pero cualquier otro valor en el parámetro X matará este sonido.

*FX211 hasta *FX214: Son llamadas que controlan el sonido generado por CTRL-G o VDU7. Son de lectura/escritura. *FX211,n controla el canal del chip de sonido en el que se genera el sonido VDU7. *FX12,n controla el volumen del sonido generado o el envoltorio empleado para generar éste. El volumen se codifica de la misma manera que el parámetro de amplitud en la orden SOUND del BASIC (o sea, −15 es muy potente, 0 no se oye y números positivos son números de envoltorio). El valor pasado en el registro X en la llamada FX es dado por (n−1)*8, donde n es el parámetro. *FX213,n controla el tono *(pitch)* de la nota generada por el VDU7; *FX214,n controla la duración de la nota generada y tocada por VDU7.

Tecla Escape: *FX229,n permite a la tecla Escape "desconectarse". *FX229,1 desactiva la tecla Escape y *FX229,0 le devuelve su efecto habitual.

*FX220,n permite al usuario establecer una tecla para generar el evento de Escape, siendo n el código ASCII de la tecla que se desea actúe por Escape. Así, *FX220,65 produce el evento Escape cada vez que se pulsa la tecla A.

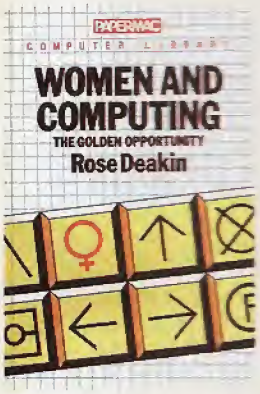
Buffers: *FX21,n disuelve, o vacía, un buffer (o memoria de paso). La operación consiste en descartar cualquier byte que se encuentre en el buffer. Por ejemplo, el vaciado del buffer del teclado desecha todas las teclas pulsadas que hayan podido acumularse mientras se ejecutaba un programa. Si usted prueba una orden GET cuando se tengan varias "teclas" sin procesar en el buffer del teclado, la orden GET tomará una tecla del buffer en vez de esperar a que se pulse una tecla. El vaciado de un buffer sonoro concluye la generación de sonido en ese determinado canal, aun habiendo otras instrucciones de sonido esperando a que se pulsen teclas. El buffer con el que se operará depende del parámetro X (véase cuadro al margen en esta página).

*FX138,n,m: Inserta el valor m en el buffer número n. Los números de los buffers son los mismos que para *FX21. Así, *FX138,0,65 introducirá la letra A en el buffer de teclado.

Parámetro X	Buffer operado por
0	Teclado
3	Impresora
4	Canal sonido 0
5	Canal sonido 1
6	Canal sonido 2
7	Canal sonido 3

Todas estas llamadas OSBYTE se pueden realizar, como es obvio, pasando los parámetros que correspondan a los registros A, X e Y y haciendo la llamada a la subrutina conforme se explicó anteriormente.

Concluimos así el estudio de las principales funciones que quedan a cargo de las llamadas OSBYTE. Habría otras más por ser examinadas detenidamente (y se pueden encontrar en el manual), pero las técnicas son las mismas que las que acabamos de describir.



"Women and computing" (Las mujeres y la informática), de Rose Deakin, Papermac, 1984

Los ordenadores transformarán la vida de las personas y de las naciones de la misma forma en que la han cambiado la rueda y el libro. Los dos libros que comentamos reflexionan, desde ópticas diferentes, acerca de esta evidencia cada vez más clara

"Women and computing"

"¿Por qué no animar a las mujeres? El director masculino es terrible; ha estado burlándose de la industria durante los últimos 20 años." Las palabras se le atribuyen a un profesor de psicología de la organización, pero tales ideas están muy ligadas al propio pensamiento de Rose Deakin, aunque ella es demasiado discreta para decirlo. Sin embargo, no pierde el tiempo deteniéndose en los hombres ni en la guerra de sexos; su preocupación es lograr que las mujeres vean la informática como una fuente de empleo y a los ordenadores como equipos industriales.

Su enfoque es tranquilo y desapasionado, presumiblemente fruto de su carrera como asistenta social, asesora de ventas de ordenadores y escritora. El libro se lee como un informe de viabilidad elaborado por un buen analista: la informática es una oportunidad que evidentemente las mujeres no

"Es posible introducirse en la informática con pocas cualificaciones y poca experiencia, y sin capacidad ni inteligencia inauditas."

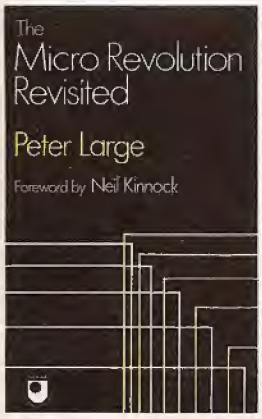
están aprovechando. Deakin considera la educación, el empleo y la promoción como los remedios a largo plazo, y la ayuda propia y la empresa como los objetivos a corto plazo; en resumidas cuentas, las mujeres han de actuar ahora para sacar provecho del nuevo mercado.

También hay, sin embargo, humor e ingenio. Deakin se complace en describir su sensación de regocijo tras dar un conferencia a 200 funcionarios públicos de gran antigüedad sobre administración de bases de datos, sólo unos pocos meses después de haber aprendido el significado del término, y está justificadamente orgullosa del libro al cual sirvió de base la conferencia.

Son de especial interés tres capítulos, en que se combinan sus virtudes personales y profesionales. Éstos describen las diferentes rutas que tomaron ella y otras siete mujeres en el campo de la informá-

"[Los investigadores señalan que]... las chicas hacen las sugerencias adecuadas para resolver el problema en cuestión; éstas son desoídas por los varones, que luego necesitan tres intentos para conseguirlo."

tica, y demuestran de forma categórica su tesis: que las aptitudes de la mujer para organización y comunicación, y su capacidad para el pensamiento lúcido y el trabajo arduo la convierten en un usuario ideal de ordenadores.



"The micro revolution revisited" (Retorno a la revolución del micro), de Peter Large, Frances Pinter, 1984

"Micro revolution revisited"

Un prólogo de Neil Kinnock, actual líder del Partido Laborista británico, el sello Set Book de la Open University de la cubierta y un corresponsal del diario *Guardian* como autor, son las credenciales de este libro. Éstas implican que la obra posee un interés social, es optimista, muy profesional y está bien documentada. El tema del libro es el desafío que representa para el *establishment* social el avance hacia la era de la informática.

"Al fin y al cabo, los ordenadores poseen una lógica matemática rígida: las personas, afortunadamente, no."

A partir de la confusa historia de los primeros 40 años de la informática, Large se detiene en los cinco peligros mortales de la informatización irreflexiva: delito, ineficacia, ignorancia, desempleo y totalitarismo. Bien podría haber añadido redundancia: este libro es, como mínimo, la sexta revisión de *The micro revolution*, escrito en 1980. Aunque comenzó proclamando con gran entusiasmo la nueva Revolución Industrial, ahora llama vigorosamente la atención sobre la posibilidad de volver a repetir los errores de la primera. Describe los artilugios y artefactos con entusiasmo y experiencia, desarrollando mientras tanto su *leitmotiv*: la vulnerabilidad de la sociedad industrial a los efectos de la informática sobre la producción, el empleo, la educación y la comunicación.

Large se muestra decididamente entusiasta respecto a las posibilidades de la tecnología, pero no sueña con que nuestra sociedad sea capaz de afron-

"Hemos dejado evolucionar nuestras máquinas en vez de volver a diseñarlas."

tar la descualificación, la pérdida de empleos y la automatización. Al comienzo del análisis del libro sobre trabajo, comunicaciones, robótica y desarrollos futuros, describe un día imaginario en la vida de Jane y Joe Babbage, alrededor de 2014 d.C. Jane edita un periódico financiero internacional desde una playa de Cornualles, mientras Joe, que es médico, realiza sus visitas rutinarias a través de la red pública de videotexto por ordenador. Los magros ingresos que obtienen con trabajos tan interesantes se comparan con el elevado salario que percibe Nat, un criado para tareas diversas de 73 años de edad que aún sabe hacer trabajos manuales. Hacia el final del libro resulta difícil saber si Large anhela o teme este futuro que imagina, y en qué cifras relativas cree él que seguiremos el cómodo camino de Jane y Joe hacia el empobrecimiento de la clase media y el sendero de Nat hacia la era de la abundancia de la clase trabajadora.